Modélisation, exploration et estimation de la consommation pour les architectures hétérogènes reconfigurables dynamiquement

Robin Bonamy

N° d'ordre : **ANNÉE 2013**

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

Ecole doctorale MATISSE

présentée par

Robin BONAMY

Préparée à l'unité de recherche IRISA / INRIA équipe CAIRN
Institut de Recherche en Informatique et Systèmes Aléatoires
ENSSAT - Université de Rennes 1

Modélisation, Exploration et Estimation de la Consommation pour les Architectures Hétérogènes Reconfigurables Dynamiquement

Thèse soutenue à l'ENSSAT - Lannion le 12 juillet 2013

devant le jury composé de :

Michel RENOVELL
Directeur de Recherche CNRS - LIRMM / *rapporteur*

Christian PIGUET
Professeur - EPFL / *rapporteur*

Christophe JEGO
Professeur des Universités - IPB/ENSEIRB / *examinateur*

Éric SENN
Maître de Conférences - Université de Bretagne Sud / *examinateur*

Olivier SENTIEYS
Professeur des Universités - Université de Rennes 1 / *examinateur*

Sébastien BILAVARN
Maître de Conférences - Université de Nice-Sophia Antipolis / *co-directeur de thèse*

Daniel CHILLET
Maître de Conférences - Université de Rennes 1 / *directeur de thèse*

Résumé

L'utilisation des accélérateurs reconfigurables, pour la conception de *system-on-chip* hétérogènes, offre des possibilités intéressantes d'augmentation des performances et de réduction de la consommation d'énergie. En effet, ces accélérateurs sont couramment utilisés en complément d'un (ou de plusieurs) processeur(s) pour permettre de décharger celui-ci (ceux-ci) des calculs intensifs et des traitements de flots de données. Le concept de reconfiguration dynamique, supporté par certains constructeurs de FPGA, permet d'envisager des systèmes beaucoup plus flexibles en offrant notamment la possibilité de séquencer temporellement l'exécution de blocs de calcul sur la même surface de silicium, réduisant alors les besoins en ressources d'exécution. Cependant, la reconfiguration dynamique n'est pas sans impact sur les performances globales du système et il est difficile d'estimer la répercussion des décisions de configuration sur la consommation d'énergie. L'objectif principal de cette thèse consiste à proposer une méthodologie d'exploration permettant d'évaluer l'impact des choix d'implémentation des différentes tâches d'une application sur un *system-on-chip* contenant une ressource reconfigurable dynamiquement, en vue d'optimiser la consommation d'énergie ou le temps d'exécution. Pour cela, nous avons établi des modèles de consommation des composants reconfigurables, en particulier les FPGAs, qui permettent d'aider le concepteur dans son *design*. À l'aide d'une méthodologie de mesure sur Virtex-5, nous montrons dans un premier temps qu'il est possible de générer des accélérateurs matériels de tailles variées ayant des performances temporelles et énergétiques diverses. Puis, afin de quantifier les coûts d'implémentation de ces accélérateurs, nous construisons trois modèles de consommation de la reconfiguration dynamique partielle. Finalement, à partir des modèles définis et des accélérateurs produits, nous développons un algorithme d'exploration des solutions d'implémentation pour un système complet. En s'appuyant sur une plate-forme de modélisation à haut niveau, celui-ci analyse les coûts d'implémentation des tâches et leur exécution sur les différentes ressources disponibles (processeur ou région configurable). Les solutions offrant les meilleures performances en fonction des contraintes de conception sont retenues pour être exploitées.

Abstract

The use of reconfigurable accelerators when designing heterogeneous system-on-chip has the potential to increase performance and reduce energy consumption. Indeed, these accelerators are commonly a adjunct to one (or more) processor(s) and unload intensive computations and treatments. The concept of dynamic reconfiguration, supported by some FPGA vendors, allows to consider more flexible systems including the ability to sequence the execution of accelerators on the same silicon area, while reducing resource requirements. However, dynamic reconfiguration may impact overall system performance and it is hard to estimate the impact of configuration decisions on energy consumption. The main objective of this thesis is to provide an exploration methodology to assess the impact of implementation choices of tasks of an application on a system-on-chip containing a dynamically reconfigurable resource, to optimize the energy consumption or the processing time. Therefore, we have established consumption models of reconfigurable components, particularly FPGAs, which assists the designer. Using a measurement methodology on Virtex-5, we first show the possibility to generate hardware accelerators of various sizes, execution time and energy consumption. Then, in order to quantify the implementation costs of these accelerators, we build three power models of the dynamic and partial reconfiguration. Finally, from these models, we develop an algorithm for the exploration of implementation and allocation possibilities for a complete system. Based on a high-level modeling platform, the implementation costs of the tasks and their performance on various resources (CPU or reconfigurable region) are analyzed. The solutions with the best characteristics, based on design constraints, are extracted.

Remerciements

Les travaux de thèse présentés dans ce mémoire et la rédaction même de celui-ci n'auraient pas pu être réalisés sans les personnes qui m'ont entouré ces dernières années.

Je tiens donc à remercier ici tous ceux qui ont contribué à la réussite de ces travaux et en particulier Daniel CHILLET, Sébastien BILAVARN et Olivier SENTIEYS pour avoir dirigé et encadré l'ensemble de ces travaux. Vos conseils, remarques et avis m'ont été très importants et constructifs.

Merci à Olivier SENTIEYS de m'avoir accueilli au sein de l'équipe projet CAIRN et merci également à toute l'équipe de Lannion. Je n'oublierai jamais les discussions socio-géopolitiques avec Karthik ou bien en électronique et logiciel avec Antoine.

Je remercie Michel AUGUIN pour m'avoir accueilli au sein du LEAT et l'ensemble de l'équipe, particulièrement soudée, qui m'a ouvert les bras.

Une pensée particulière s'adresse à ma famille pour le soutient sans faille pendant la totalité de mes études et pour me permettre de me changer d'idées pendant les vacances.

Je tiens bien évidemment à remercier Romain pour avoir partagé l'appartement pendant mes études à Lannion. Nous avons réalisé beaucoup de projets, principalement en électronique et le partage de cette passion était formidable. Merci également d'avoir relu ce mémoire.

Merci à Mathieu pour l'ensemble des remarques, conseils et partages au cours de cette thèse. Merci également pour les multiples randonnées inoubliables dans les Alpes, que l'on poursuivra sans doute. Je n'oublie pas ton œil complètement extérieur sur la relecture de ce document.

J'adresse toute ma reconnaissance aux membres du jury, Michel RENOVELL et Christian PIGUET pour l'intérêt porté sur mes travaux en acceptant d'être rapporteurs, Christophe JEGO et Éric SENN pour l'évaluation de ma thèse.

Mes remerciements vont également à l'ensemble des partenaires du projet OpenPEOPLE, projet dans lequel s'intègrent mes travaux, pour avoir échangé sur nos domaines proches, orientés sur la consommation d'énergie. Ce projet n'aurait pas pu voir le jour sans le financement de l'Agence Nationale de la Recherche et du pôle de compétitivité Images et Réseaux.

« Only wimps use tape backup :
real men just upload their
important stuff on ftp, and let the
rest of the world mirror it ;) »

Linus TORVALDS

Table des matières

Table des figures

Liste des tableaux

Introduction

L'ENIAC (*Electronic Numerical Integrator Analyser and Computer*), présenté en 1946, est le premier ordinateur à fonctionnement entièrement électronique. Son architecture était basée sur plus de 17 000 tubes à vides et permettait de réaliser près de 100 000 additions par seconde. L'ENIAC occupait une salle de 170 m^2 et la consommation était de 150 kW, les tubes à vide devant être chauffés pour atteindre la température de fonctionnement adéquate. L'efficacité énergétique était alors de 0.6 OPS/W - Opération Par Seconde par Watt. La programmation des opérations s'effectuait par un câblage manuel. De plus, les tubes à vide tombaient régulièrement en panne et la durée de fonctionnement sans interruption ne dépassait pas quelques jours.

Depuis l'invention du premier ordinateur, la technologie a beaucoup évolué. La mise au point du transistor en 1947 a permis de remplacer avantageusement le tube à vide. Ces transistors, bipolaires, sont plus petits et plus fiables. À la fin des années 50, les premiers circuits intégrés apparaissent : plusieurs transistors sont placés dans le même circuit ce qui permet d'améliorer le taux d'intégration. Puis le transistor à effet de champ apparaît sous la forme de MOSFET. Celui-ci supplante rapidement la technologie bipolaire grâce à son fonctionnement sur niveaux de tension ce qui réduit la consommation. Les techniques de réalisation des MOSFETs et des circuits intégrés s'améliorent rapidement et favorisent leur intégration. En 1965, Gordon Earle Moore prédit que la complexité des circuits intégrés doublerait tous les ans. Réévalué en 1975 pour porter sur le nombre de transistors dans les processeurs, ce postulat, appelé Loi de Moore, s'est avéré proche de la réalité. Aujourd'hui, un processeur multicœur en contient plus d'un milliard et 100 milliards d'instructions par seconde peuvent être effectuées avec une consommation de 170 W (soit une efficacité énergétique de 600 $MIPS/W$ - Instructions Par Seconde par Watt). Les performances sont bien supérieures aux premiers processeurs, cependant la densité engendre des problèmes de consommation électrique (énergie) et de dissipation thermique.

Ces problèmes de consommation sont d'autant plus importants pour une classe particulière de systèmes : les systèmes dits embarqués. Ce sont des produits électroniques généralement conçus pour une application spécifique et contraints par leur environnement. Ils sont présents dans toutes les applications autonomes, dans les satellites, les réseaux de capteurs, les appareils mobiles, les véhicules... Ces sys-

tèmes embarqués ont un certain nombre de contraintes liées à la taille du circuit, à la consommation d'énergie, au coût de fabrication et au temps d'exécution principalement. Parmi ces contraintes, la contrainte énergétique est cruciale. En effet, nombre de ces systèmes sont autonomes et alimentés soit par une batterie dont la durée de fonctionnement doit être suffisamment longue, soit grâce à des systèmes de récupération d'énergie (cellules photovoltaïques notamment) dont la production d'électricité est limitée. De plus, les problèmes liés à la dissipation de l'énergie et à la hausse de température peuvent endommager définitivement le circuit ou altérer temporairement son fonctionnement. De manière générale, la consommation intervient directement sur l'autonomie des systèmes embarqués autonomes, et cette longévité est un enjeu économique et environnemental. Les systèmes embarqués étant de plus en plus répandus, il est essentiel de concentrer les efforts sur la réduction de la consommation.

Les architectures pour le calcul embarqué ont été conçues et optimisées afin de réduire la consommation. Des processeurs ayant des performances plus faibles mais une meilleure efficacité énergétique (de l'ordre de 1 200 $MIPS/W$ ou plus) sont utilisés. Certains systèmes intègrent plusieurs cœurs de processeurs du même type (SMP, architectures homogènes) ou des unités de calcul de différentes natures (architectures hétérogènes) pour améliorer les performances et/ou l'efficacité énergétique. Parmi ces unités de calcul figurent les circuits matériels dédiés (ASIC) qui implémentent les traitements spécifiques et ont une efficacité énergétique de l'ordre de 100 000 $MOPS/W$. Ces accélérateurs matériels ont cependant un inconvénient majeur, ils sont statiques. Contrairement à un code s'exécutant sur un processeur, la fonctionnalité d'un accélérateur ne peut pas être modifiée. De plus, ce type de solution monopolise une surface de silicium dont le taux d'utilisation est parfois relativement faible. Cette surface a un impact sur le coût et la consommation du circuit. C'est pour apporter de la flexibilité dans la conception et l'utilisation des accélérateurs matériels que les architectures reconfigurables ont été introduites dans les années 80. Ce type d'architecture permet de configurer des traitements à réaliser à partir de fonctions logiques élémentaires programmables. Contrairement aux ASICs, cette configuration n'est pas figée : il est ici possible de modifier les traitements réalisés par une opération de téléchargement dans une région du système. C'est la reconfiguration.

Le FPGA (*Field-Programmable Gate Array*) est un type de circuit reconfigurable qui intègre généralement plusieurs ressources (blocs logiques, bascules, mémoires, circuits d'horloge, processeurs...). Depuis les années 2000, certains de ces circuits sont reconfigurables partiellement et dynamiquement, il est alors possible de modifier la configuration d'une partie du circuit sans empêcher le fonctionnement du reste

du composant.

La reconfiguration dynamique rend les systèmes plus flexibles. Elle permet d'exploiter des accélérateurs matériels performants qui peuvent être mis à jour sans interrompre le fonctionnement du circuit. Un accélérateur peut être configuré uniquement en cas de besoin. Lorsqu'il n'est plus utilisé, la zone où il était instancié est disponible pour d'autres accélérateurs, ce qui contribue à une meilleure utilisation de la zone reconfigurable et à la réduction de la surface globale du système. Par ailleurs, d'autres techniques liées à la reconfiguration dynamique peuvent être employées pour réduire davantage la consommation. La consommation énergétique étant une contrainte forte des systèmes embarqués, l'utilisation d'accélérateurs matériels associée à la reconfiguration dynamique est avantageuse.

Si les perspectives de réduction de la consommation sont prometteuses, l'exploitation de ce potentiel pose des défis importants. Il est aujourd'hui difficile d'évaluer l'impact de la reconfiguration sur le système global.

À ce titre, l'utilisation de la reconfiguration dynamique des accélérateurs matériels est une technique à considérer au plus tôt dans les phases de conception d'un système, en vue de réduire la consommation énergétique. Les approches pour la conception à partir de niveaux d'abstraction élevés permettent de répondre à ces contraintes et requièrent une estimation de la consommation énergétique. Cependant, l'intégration d'une ressource reconfigurable, en particulier avec de la reconfiguration dynamique, est plus complexe que l'emploi d'un accélérateur matériel dédié, statique, et nécessite une modélisation spécifique de sa consommation. Dans ce contexte, l'objectif principal de cette thèse est de montrer dans quelle mesure la reconfiguration dynamique permet de réduire la consommation dans les architectures hétérogènes reconfigurables.

Les travaux de cette thèse proposent une formalisation de l'utilisation de la reconfiguration dynamique dans un système hétérogène. Une méthodologie d'exploration basée sur cette formalisation analyse l'implémentation, l'ordonnancement des tâches et l'allocation des ressources dans un système hétérogène multiprocesseur reconfigurable dynamiquement. Les résultats de l'exploration se présentent sous la forme de compromis performance-consommation qu'il est possible de comparer avec des solutions caractéristiques (statiques, logicielles). La seconde contribution majeure est l'étude de consommation lors de la reconfiguration dynamique. Trois modèles sont proposés en fonction : i) du niveau d'utilisation de l'estimation, ii) de la complexité de mise en place et iii) de la précision de l'estimation. Ces modèles permettent d'évaluer précisément l'impact de la reconfiguration sur la consommation et contribuent aux choix d'implémentation des tâches. Finalement, un modèle de consommation

énergétique en fonction de l'exploitation du parallélisme est étudié. Le modèle obtenu est destiné à être utilisé pour la caractérisation à haut niveau de solutions matérielles et à offrir plusieurs instances de tâches avec différents compromis de performance, de consommation et de surface.

Ces contributions sont présentées dans ce mémoire et organisées en quatre chapitres. Le chapitre 1 présente la consommation dans les circuits et les méthodes possibles pour son estimation et sa réduction. La suite de l'état de l'art se concentre sur la consommation dans les architectures reconfigurables et l'impact de la reconfiguration dynamique est présenté. Finalement, la modélisation et l'abstraction haut niveau des systèmes électroniques sont abordées. Le chapitre 2 est consacré à l'étude de l'influence du niveau de parallélisme sur la consommation énergétique des accélérateurs matériels. Un modèle de consommation d'énergie en fonction de l'accélération (obtenue grâce à l'augmentation du parallélisme) est construit. L'augmentation du parallélisme accroît la surface de l'accélérateur ce qui a un impact sur le coût de la reconfiguration dynamique. Une formalisation tenant compte de toutes les puissances en jeu lors de l'utilisation de la reconfiguration dynamique des accélérateurs matériels dans un système est proposée. Le chapitre 3 s'intéresse quant à lui à la modélisation de la consommation du processus de reconfiguration dynamique partielle. Basés sur une analyse très précise des mesures, trois modèles sont développés pour permettre l'estimation à plusieurs niveaux de précision et de complexité d'utilisation. Le chapitre 4 présente ensuite une méthode qui s'appuie sur ces modèles pour explorer et estimer la consommation dans les systèmes reconfigurables dynamiquement. Cette méthode permet d'identifier une (ou plusieurs) solution(s) d'ordonnancement des tâches et d'allocation des ressources qui minimise(nt) l'énergie ou le temps d'exécution. Les résultats de cette méthode d'exploration mettent en évidence certaines conditions pour garantir que les gains engendrés par la reconfiguration en excèdent ses coûts.

Ces travaux se sont inscrits dans le cadre du projet Open-PEOPLE (*Open-Power and Energy Optimization PLatform and Estimator*), porté par l'ANR (Agence Nationale de la Recherche). L'objectif du projet, qui s'est terminé en décembre 2012, était de proposer une plateforme ouverte et accessible à distance pour l'estimation et l'optimisation de la consommation. La modélisation sur laquelle repose la méthodologie d'exploration est basée sur une description multi-niveaux, supportée par le langage AADL (*Architecture Analysis & Design Language*) et développée par les partenaires du projet. Associée à des modèles de consommation énergétique, l'approche à plusieurs niveaux peut supporter l'estimation de la consommation à chaque

étape de la conception et la mise en place des techniques d'analyse et d'optimisation de la consommation.

La plateforme Open-PEOPLE cible les architectures hétérogènes et reconfigurables. Les travaux présentés dans ce mémoire ont été réalisés dans le cadre de ce projet et abordent les points particuliers de l'estimation, de la modélisation et de l'analyse de la consommation pour des systèmes qui intègrent la possibilité de reconfiguration dynamique. Ces travaux ont été menés conjointement par l'équipe CAIRN de l'IRISA et le LEAT dont les axes de recherche se situent en particulier sur les architectures reconfigurables, les systèmes embarqués et les problématiques de réduction de la consommation.

1 État de l'art

Contenu

CMOS pour **C**omplementary **M**etal-**O**xide-**S**emiconductor, traduit litéralement par semi-conducteurs à oxydes métalliques complémentaires, est une technologie de fabrication des circuits intégrés. Brevetée en 1967, la technologie CMOS est aujourd'hui utilisée pour un grand nombre de composants électroniques numériques tels que les circuits logiques, les microprocesseurs et microcontrôleurs, les mémoires mais aussi pour les composants analogiques, notamment les capteurs photographiques et les convertisseurs de signaux (convertisseurs analogique-numérique et inversement). La consommation électrique est une contrainte majeure dans la plupart des composants et en particulier pour les systèmes embarqués. Pour cette raison, la consommation des circuits intégrés est une préoccupation de longue date et est évoquée dans la première partie de ce chapitre. La seconde section introduit les circuits reconfigurables qui apportent une flexibilité de conception et d'utilisation des accélérateurs matériels. La consommation dans ce type de circuit est présentée et en particulier le coût de la reconfiguration dynamique. Finalement, la modélisation à haut niveau des architectures reconfigurables pour la prise en compte de la consommation est présentée et la problématique issue de cet état est présentée.

1.1 Consommation dans les circuits

1.1.1 Consommation dans les circuits CMOS

Il existe deux types de transistor en fonction de leur fonctionnement sur l'alimentation (Vdd) ou sur la masse (Vss), respectivement P et N. Ces deux transistors sont

placés symétriquement entre l'alimentation et la masse tel que présenté Figure 1.1 ce qui constitue la fonction de base réalisée par des transistors CMOS, un inverseur de signal. Ceci permet, en fonction du niveau de tension appliqué sur la grille des transistors, d'avoir soit le transistor de type N passant, soit le transistor de type P passant. Lorsque l'entrée est à l'état 1 (niveau haut), le transistor N est passant et le transistor P est bloqué, alors la sortie est à l'état 0 (niveau bas). Le fonctionnement est inversé si l'entrée est à 0.

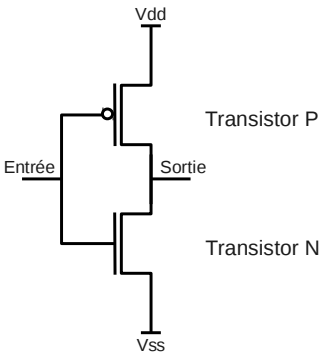

FIGURE 1.1: Un inverseur, fonction de base réalisée par des transistors CMOS.

À partir de cette fonction de base, il est possible de réaliser des fonctions logiques telles que des portes NAND (NON-ET) ou des bascules. La technologie CMOS est alors utilisée dans la plupart des circuits intégrés, cependant, son fonctionnement produit des pertes de courant. Ces pertes se traduisent par une consommation d'énergie et par un échauffement du circuit. La consommation d'un inverseur CMOS (et de toute autre fonction) peut être regroupée en deux familles. Il s'agit de la consommation dynamique et la consommation statique. Les principes de base de la consommation dans les circuits CMOS sont exprimés par la suite, la formulation détaillée est exposée dans le livre [3] ainsi que la plupart des concepts liés à la consommation.

1.1.1.1 Consommation dynamique

La structure CMOS, qui consiste à placer un (ou plusieurs) transistor(s) de type P entre l'alimentation et la sortie (afin de pouvoir générer un signal à l'état 1) et un (ou plusieurs) transistor(s) N entre la sortie et la masse (pour générer un signal à l'état 0), est une structure qui peut provoquer une fuite de courant via ces transistors. Les grilles des transistors forment une surface possédant des charges, celle-ci se comporte alors comme une capacité par rapport aux autres éléments du transistor et du circuit intégré. La grille a donc besoin d'un courant de charge pour passer au niveau haut. Ce courant est issu de l'alimentation de l'étage en amont commandant le transistor. Puis lorsque cette grille doit passer au niveau bas, le courant de décharge est envoyé vers la masse. Tous les deux changements d'état

d'un étage CMOS, la quantité d'énergie emmagasinée par la capacité de la grille est perdue (Figure 1.2). En plus de la capacité de la grille, les capacités des connexions entre les étages se comportent de la même manière que la capacité de grille ce qui augmente le courant de commutation.

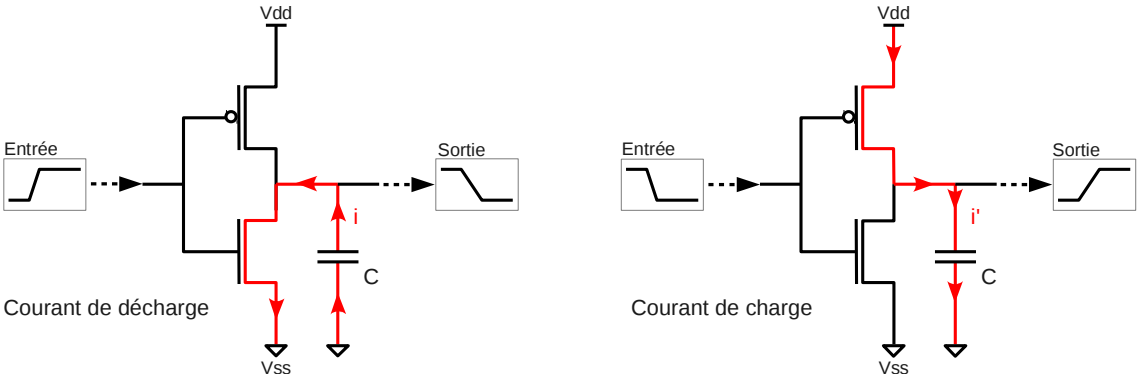

FIGURE 1.2: Courant de commutation dans un étage inverseur CMOS.

La consommation dynamique est alors donnée par l'équation 1.1 où V_{dd} représente la tension d'alimentation du circuit, F la fréquence des transitions, α le taux de transitions réelles par rapport à la fréquence de base et C la totalité des capacités parasites.

$$P_{dyn} = {V_{dd}}^2 \times F \times \alpha \times C \tag{1.1}$$

Compte tenu de l'intervention de la tension d'alimentation au carré dans cette puissance dynamique, il s'agit du premier paramètre sur lequel les concepteurs agissent pour diminuer la consommation. Cependant, la baisse de la tension ralentit la commutation des transistors et réduit la fréquence de fonctionnement, c'est une technique de réduction de la consommation, présentée dans la section 1.1.3.3.

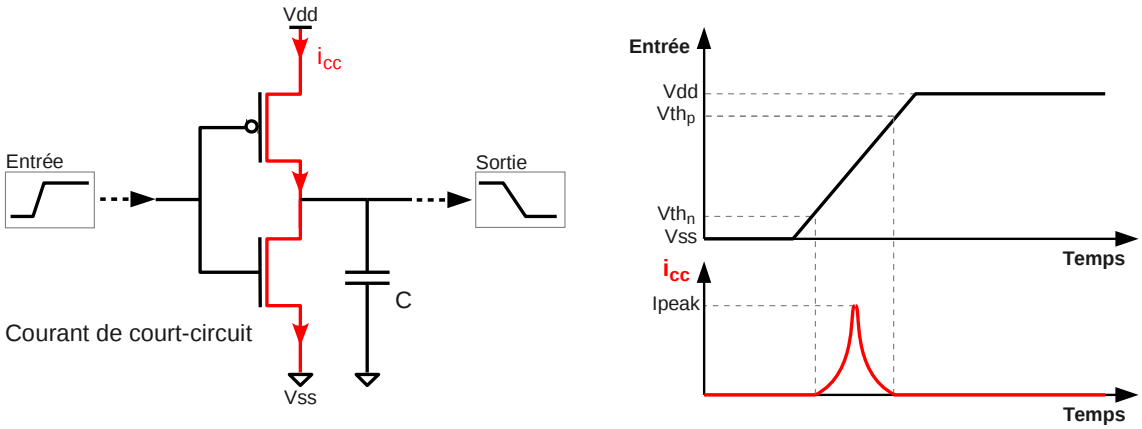

FIGURE 1.3: Courant de court-circuit dans un étage inverseur CMOS.

De plus, lors d'un changement d'état, un transistor devient passant tandis que le transistor complémentaire devient bloquant. Durant cette transition, les deux transistors sont passants pendant un court laps de temps. Ils se comportent alors comme une résistance entre l'alimentation et la masse pendant leur changement d'état, provoquant un courant de court-circuit I_{cc}, représenté Figure 1.3. La consommation de court-circuit est une équation complexe dépendant des paramètres liés à la technologie de réalisation des transistors, notamment les tensions de seuils de commutation des transistors (Vth_P et Vth_N) et le courant de court-circuit maximum I_{peak}. Une écriture simplifiée est donnée par l'équation 1.2 et cette puissance s'ajoute à P_{dyn}.

$$P_{SC} = V_{dd} \times f(F, \alpha, I_{peak}, Vth_P, Vth_N) \times N_{tr} \tag{1.2}$$

1.1.1.2 Consommation statique

Il existe cependant une consommation même lorsque le circuit n'effectue plus de transitions, lorsque la fréquence est nulle ou lorsqu'il n'y a pas d'activité ($\alpha = 0$). Lorsque les transistors sont ouverts, leur résistance est très élevée, mais pas nulle. Il en résulte un courant de fuite I_{leak} qui traverse les transistors complémentaires entre l'alimentation et la masse, représenté Figure 1.4. Cette consommation est directement liée au nombre de transistors N_{tr} (surface du circuit) et à la tension d'alimentation V_{dd}, elle est donnée par l'équation 1.3.

$$P_s = V_{dd} \times I_{leak} \times N_{tr} \tag{1.3}$$

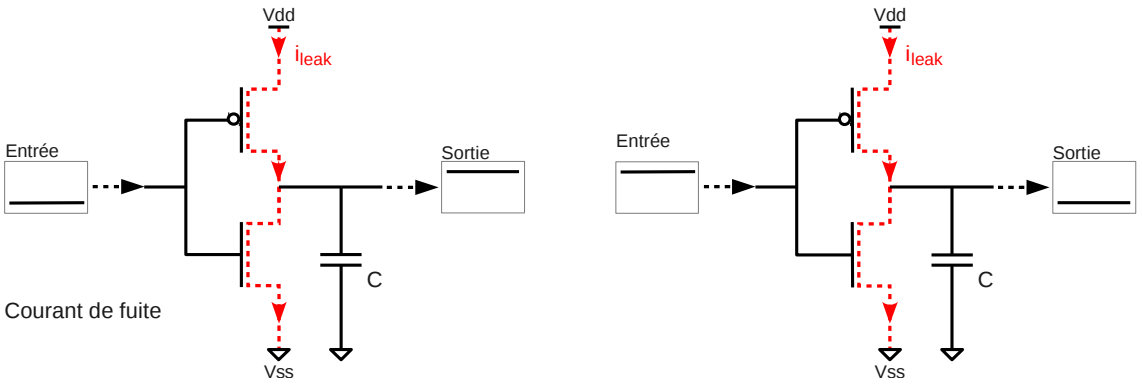

FIGURE 1.4: Courant de fuite dans un étage inverseur CMOS.

Pour réduire la puissance statique, la tension peut être abaissée ou l'alimentation coupée si le circuit n'est pas utilisé (technique décrite dans la section 1.1.3.2). La technologie de réalisation du circuit intervient sur les fuites des transistors.

1.1.1.3 Évolution de la technologie

Selon la seconde loi de Moore, le nombre de transistors sur un circuit intégré double tous les deux ans, soit une croissance exponentielle [4]. Cette prédiction (1975) s'est avérée assez exacte et suivie notamment par les concepteurs de processeurs. Le nombre de transistors intégrés dans les processeurs en fonction du temps est représenté Figure 1.5. Cette loi a été suivie notamment grâce à la réduction de largeur

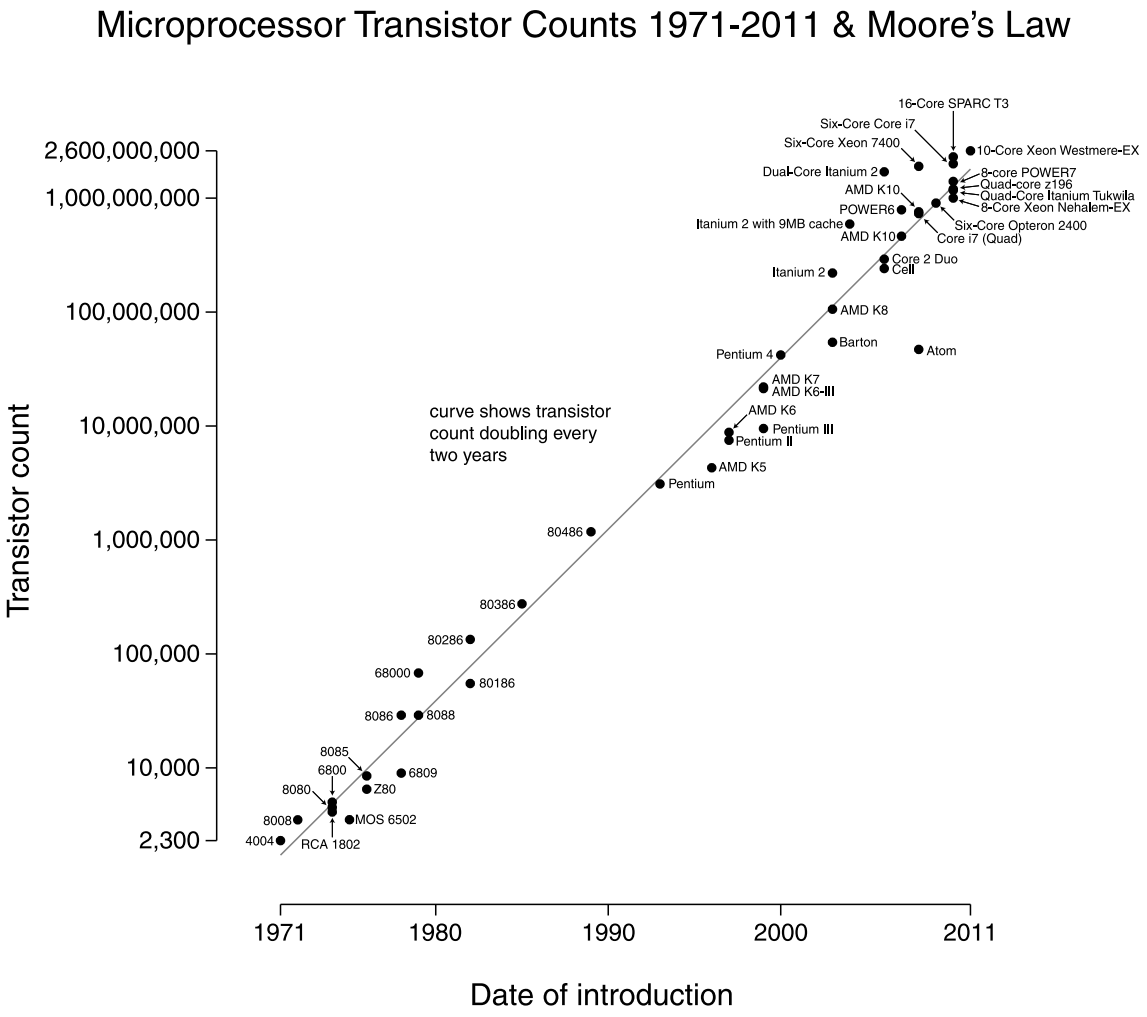

FIGURE 1.5: Le nombre de transistors dans les processeurs suit une croissance exponentielle.
©Wgsimon, wikimedia, CC-BY-SA 3.0

de gravure des transistors. Le processeur Intel 4004 lancé en 1971, par exemple, était gravé en 10 μm. Aujourd'hui, en 2013, Intel réalise ses processeurs en 22 nm [5] et est sur la voie du 14 nm. La réduction de largeur de gravure permet de concevoir des transistors plus petits et favorise une intégration plus dense. La réduction de largeur des transistors diminue ses capacités parasites, puisque sa surface est plus faible, et donc de réduire la consommation dynamique pour un même nombre de

transistor. Cependant la réduction de cette largeur provoque un passage du courant de fuite plus important entre l'alimentation et la masse au travers des éléments du circuit. Ceci engendre une augmentation de la consommation statique qui devient prédominante dans les circuits à largeur de gravure fine [6].

1.1.2 Méthodes d'estimation

Les formules classiques (puissance statique équation 1.3 et dynamique équation 1.1) peuvent être utilisées pour l'estimation de la consommation. À première vue, ces équations sont simples mais en réalité leur application est plus complexe. Premièrement, elles correspondent à un modèle théorique de base. La température par exemple joue un rôle non négligeable dans la consommation puisque sa contribution au niveau de la puissance statique est assimilable à une inférence exponentielle [7]. Deuxièmement, il s'agit d'équations à bas niveau et il faut avoir connaissance des paramètres notamment de surface (N_{tr}) et de taux d'activité des circuits (α). De plus, l'équation de consommation de puissance dynamique est établie au niveau transistor avec connaissance de l'activité du signal et la capacité parasite propre à ce signal.

Cette section regroupe des méthodes d'estimation de la consommation et les améliorations proposées afin d'améliorer la précision des estimations en présentant tout d'abord les travaux concernant le bas niveau (niveau transistor ou RTL) puis en relevant progressivement le niveau de l'estimation en passant par la caractérisation de blocs élémentaires pour atteindre l'estimation au niveau de l'algorithme ou de l'application.

1.1.2.1 Bas niveau - Portes logiques

L'estimation de la consommation à bas niveau est en général complexe car tous les paramètres de chaque transistor ou fonction élémentaire sont pris en compte. Pour obtenir une estimation des transitions effectuées par les transistors au sein d'un bloc de traitement dont on veut estimer la consommation, les analyses sont souvent effectuées à partir de simulations [8], soit à partir d'un vecteur de test, soit à partir d'analyses probabilistes et statistiques afin d'analyser la propagation des transitions [9]. Cependant même à très bas niveau, ces estimations sont perfectibles. Par exemple [10] propose de prendre en compte les *glitches* puisque ces variations de signaux transitoires ont un effet sur la consommation. Avec l'estimateur développé, l'erreur d'estimation est environ divisée par trois entre une estimation avec prise en compte des *glitches* comparée à l'estimation sans prise en compte.

La complexité croissante des applications et le détail nécessaire pour l'estimation à bas niveau apportent le besoin d'une simplification des estimateurs tout en

conservant une bonne qualité d'estimation de la consommation. En ce sens, plusieurs contributions existent, par exemple dans [11, 12] les auteurs proposent un modèle de consommation au niveau RTL basé uniquement sur les informations structurelles, c'est-à-dire principalement les capacités des connexions. Ce modèle de consommation est alors indépendant des statistiques des signaux. Ceci permet d'éviter l'usage de simulations avec vecteurs de données ou simulations probabilistes et donc de réduire le temps d'estimation tout en permettant une estimation environ dix fois plus précise que l'utilisation de modèles dont les vecteurs de données sont caractérisées. De plus, un avantage intéressant proposé dans cette méthode est la possibilité d'estimer un résultat pire cas, ce qui permet de dimensionner les alimentations et le refroidissement d'un système. Ces fonctionnalités sont intégrées, parmi d'autres non décrites ici, dans l'outil d'estimation RTLEst [13].

1.1.2.2 Caractérisation de blocs

Pour élever un peu le niveau de l'estimation, celle-ci peut s'effectuer par le biais de blocs élémentaires dont la consommation est déjà caractérisée. Ainsi l'estimation globale de la consommation peut s'appuyer sur des blocs élémentaires pré-caractérisés à partir du taux de commutation moyen des entrées du bloc [14]. Ces blocs sont principalement les registres, les multiplexeurs et les portes logiques. En plus, une modélisation des inter-connexions et de l'arbre d'horloge à partir du nombre de blocs connectés permet d'améliorer la précision de l'estimation. Cette approche obtient une bonne précision (8% en moyenne pour les blocs et 20% pour les connexions) et elle est beaucoup plus rapide qu'une estimation par simulation classique (jusqu'à 38 fois plus rapide). Dans [15], les auteurs proposent une spécification plus poussée des vecteurs d'entrée des blocs. Ils caractérisent les données suivant deux types de bits, le bit de poids fort qui correspond au signe et les bits de poids faible. Ceci part du constat que les opérateurs n'ont pas la même consommation en fonction des variations du bit de signe comparé aux autres bits. Les blocs élémentaires sont caractérisés en capacité par simulation bas niveau en fonction du type de bit (bit de signe, bits de poids fort et de poids faible) et des transitions des vecteurs d'entrée. Cette méthode permet une estimation relativement précise ($10-15\%$), mais nécessite une caractérisation de la capacité des blocs à étudier et donc toujours d'obtenir les informations d'implémentation bas niveau. Cette méthode d'estimation considérant le poids des bits est reprise par [16] pour le traitement du signal. La modélisation de chaque type d'éléments d'un système (opérateurs, interconnexions et mémoires) est effectuée. L'estimation de la consommation du système nécessite la construction de modèles analytiques de propagation du signal dans les différents blocs et l'erreur d'estimation obtenue est inférieure à 15%.

Il est possible de caractériser directement la consommation dans une table en trois dimensions dont les paramètres sont la densité de probabilité du signal d'entrée, le taux de transition de ce même signal et le taux de transition en sortie. La sortie est calculée en *zero delay*, c'est-à-dire en considérant que le bloc est en logique pure et qu'il n'y a pas de retards et de *glitches*. Lors de la caractérisation du bloc en faisant varier ces paramètres, la valeur de puissance consommée est calculée à partir d'une simulation de Monte Carlo et inscrite dans la table [17]. Lorsque la valeur réelle de l'une au moins des trois entrées de la table n'existe pas, la puissance est estimée par interpolation. Cette méthode permet d'obtenir des estimations avec des erreurs de l'ordre de 6%. Cependant cette méthode ne permet pas de considérer tous les cas de densité de transition des entrées et la précision de l'estimation est impactée [18]. Pour palier ce problème, [19] propose une table avec *cluster* ce qui permet notamment de favoriser l'estimation de la puissance des blocs dont les vecteurs d'entrée ne sont pas aléatoires, mais biaisés. La méthode de *clusterisation* consiste à caractériser la table non plus selon trois entrées mais sur la base de six entrées. Les paramètres des densités de probabilités du vecteur d'entrée et les probabilités de transition en entrée et en sortie sont découpés en deux, les bits avec forte probabilité de transition et ceux avec faible probabilité. Les bits de données et de contrôle sont séparés ce qui permet de mieux estimer la puissance lorsque les signaux ne sont pas complètement aléatoires. Là où la table à trois dimensions classique obtient une erreur d'estimation moyenne de 23%, cette méthode permet de réduire l'erreur à 5%. [20] ainsi que [21] proposent le même type d'approche sauf que l'estimation est effectuée à partir d'une équation à quatre paramètres : la densité de probabilité du signal d'entrée, la probabilité de transition en entrée, un coefficient de corrélation spatiale des vecteurs d'entrée et la probabilité de transition en sortie. Les paramètres de l'équation sont déterminés par un algorithme récursif de minimisation de l'erreur quadratique. De cette manière, lors de l'estimation, l'erreur est de 10% en moyenne. [22] présente une méthode de caractérisation plus rapide se situant entre la table à trois dimensions et le modèle analytique. La solution proposée construit une table 3D de manière classique, mais intègre aussi une équation d'interpolation de la puissance consommée selon les trois paramètres de la table pour chaque point caractérisé. Le nombre de points de la table est réduit grâce à l'interpolation obtenue par l'équation et ajustée en fonction de la position dans la table. Cette méthode obtient une diminution du temps d'estimation proche d'un facteur 7 et une d'erreur d'estimation autour de 5%.

Une autre méthode proposée pour estimer la puissance consommée par un bloc est d'utiliser la différence entre deux vecteurs d'entrée consécutifs. La différence entre deux vecteurs est caractérisée par la distance de Hamming [23]. Cette mesure corres-

pond au nombre de bits qui diffèrent entre deux vecteurs. La distance alors calculée en moyenne est caractérisée en fonction de la puissance consommée pour permettre son utilisation en tant qu'estimateur. Cette méthode permet une estimation avec une erreur de l'ordre de 15 à 20 % en général, mais peut s'accroître en fonction du biais du vecteur d'entrée. Ce type d'estimation est mieux adapté pour les blocs à traitement de flux de données important.

Les méthodes d'estimation de la consommation par caractérisation de blocs vues ci-dessus ont chacune leur avantage, mais aussi des limites notamment sur la nature aléatoire des signaux d'entrée. Afin de sélectionner le meilleur modèle, [24] propose une méthode d'estimation à plusieurs modèles. La pertinence des modèles est sélectionnée en fonction du jeu de vecteurs d'entrée et de l'erreur obtenue lors d'une phase de caractérisation. Cette méthode, au coût d'une très faible augmentation du temps d'estimation, permet d'obtenir des erreurs bien inférieures (7 fois) à la moyenne des méthodes à modèle unique. Cependant, si le temps de la phase d'estimation n'est que très peu impacté par cette approche multi-modèles, le temps de caractérisation existe lui toujours et est forcément plus important lors de l'utilisation de plusieurs modèles.

1.1.2.3 Haut niveau - Algorithmique

Ces méthodes d'estimation s'appliquant sur des blocs et macro-blocs s'éloignent doucement de la modélisation bas niveau pour atteindre un niveau d'abstraction supérieur. Elles nécessitent une bonne connaissance du niveau matériel pour être précises, notamment en passant par une étape de caractérisation spécifique. La modélisation de haut niveau cherche à s'affranchir au maximum des variations et paramètres liés à la mise en œuvre du bloc à caractériser et se base sur les paramètres disponibles lors des premières phases de conception. Il s'agit principalement : (i) des paramètres du système, tels que la taille du circuit (ii) des paramètres et contraintes algorithmiques tels que le délai, débit voulu, le niveau de complexité de l'algorithme par exemple l'ordre d'un filtre ou la taille de la FFT permettent d'obtenir le nombre d'opérations à effectuer, les accès mémoires et complexité du contrôle (iii) des paramètres architecturaux comme la fréquence d'horloge, le type et l'accès à la mémoire et les paramètres technologiques, notamment le type du circuit et sa technologie de conception. La modélisation haut niveau est en général effectuée sur un algorithme. Cependant un algorithme accéléré matériellement est découpé en deux parties, la partie de traitement des données et la partie de contrôle qui assure notamment la synchronisation des opérateurs avec les accès mémoire pour effectuer le calcul demandé. La puissance consommée par la partie de traitement des données dépend des

opérateurs et vecteurs de données et peut être estimée selon les méthodes de macro-bloc précédemment décrites alors que la puissance consommée par la partie contrôle de l'algorithme est en général provoquée par l'activité d'une machine d'états. Un moyen d'améliorer la précision de l'estimation passe donc par une modélisation de la machine d'états sous la forme d'un graphe afin d'estimer les probabilités de transitions [25]. L'estimation de la puissance consommée se base sur la fréquence d'horloge et les probabilités de changer d'état lors d'une période. L'estimation obtenue conduit à une erreur d'environ 25%. De la même manière, [26] propose la modélisation d'une machine selon un graphe état/transition représenté par un *n-uplet* comprenant les entrées et les sorties, l'état courant et l'état suivant. L'estimation proposée est effectuée à partir des paramètres statiques de comportement (le nombre d'entrées, de sorties, nombre d'états, nombre de n-uplets), des paramètres dynamiques de comportement (probabilité moyenne des signaux d'entrée, de transition d'entrée, de transition de sortie et de changement d'état) et des paramètres statiques structurels (le nombre d'états et de variables d'états). Cette méthode d'estimation nécessite un niveau de détail assez élevé des paramètres de la machine d'états, l'erreur d'estimation est de 34%.

Dans [27], les auteurs proposent une méthodologie d'estimation en utilisant seulement les paramètres du système, de l'algorithme, de l'architecture et de la technologie qui sont disponibles. En réalité, il s'agit de plusieurs méthodes d'estimation de la consommation à plusieurs niveaux, ce qui permet un raffinement de l'estimation en fonction des paramètres de conception disponibles. L'estimation à haut niveau (niveau algorithmique) consiste à transcrire dans ce cas, un code algorithmique en un code implémentable (bas niveau) et à effectuer l'estimation sur ce dernier. Les modèles de consommation sont construits à partir d'un découpage en blocs du système et une analyse du rôle de ces paramètres fonctionnels dans la consommation. En fonction du niveau d'estimation et du nombre de paramètres connus, notamment le taux d'activité, l'erreur d'estimation se situe entre 3 et 30 %.

De manière générale, l'estimation de la consommation au sein des circuits intégrés fait l'objet d'un compromis entre la précision de l'estimation et le niveau d'abstraction. Les systèmes devenant de plus en plus complexes, il n'est pas possible d'effectuer des simulations bas niveau très précises, principalement pour des raisons de complexité et de temps. La consommation d'énergie est une problématique principale des systèmes embarqués et l'estimation à haut niveau facilite la conception de système en permettant une considération de la consommation au plus tôt. Les méthodes présentées ici abordent les principales techniques d'estimation de la consommation dans différentes situations. En fonction des paramètres connus, la

précision de l'estimation varie de 8% à 34% en moyenne. Les méthodes présentées sont basées sur une phase de caractérisation qui peut être fastidieuse.

La section suivante traite les techniques de réduction de la consommation dans les circuits CMOS en général, puis la consommation dans les circuits configurables, notamment les FPGAs, est abordée.

1.1.3 Méthodes de réduction

Comme mentionné dans la section précédente, la tension d'alimentation, la fréquence de fonctionnement, le taux d'activité ainsi que la surface sont les paramètres déterminants dans la consommation d'un circuit intégré. Les méthodes présentées ci-dessous permettent d'adapter ces paramètres dans le but de réduire la consommation en puissance ou en énergie, en fonction des besoins de l'application.

1.1.3.1 Clock gating

Le *clock gating* est une technique consistant à stopper la propagation de l'horloge dans une zone spécifique du circuit intégré lorsque celle-ci n'est plus utilisée. Ceci permet de stopper toute activité dans la zone concernée et donc d'annuler la consommation dynamique associée. Cependant, l'ajout d'un bloc de gestion de l'horloge augmente la capacité de l'arbre d'horloge et introduit des délais [28]. La consommation statique reste identique. Par exemple dans [29], les auteurs appliquent cette technique sur des machines d'états. Les machines d'états attendent généralement des évènements, pendant ce temps l'horloge continue de fonctionner dans les bascules et la mise au point d'une gestion du *clock gating* a permis une réduction de la consommation estimée à 25%.

1.1.3.2 Power gating

Le power gating est une technique qui consiste quant à elle à couper toute l'alimentation d'une zone spécifique du circuit lorsque qu'elle n'est plus utilisée. Cette technique permet de stopper la consommation statique et dynamique hormis quelques fuites de courant au niveau des interfaces. La conception est assez complexe et nécessite une isolation électrique de la zone à alimenter par des cellules spécifiques, ce qui provoque une perte de place significative [30]. D'autres problèmes liés à la coupure d'alimentation apparaissent tels que la conservation des données en mémoire. Il est alors nécessaire de sauvegarder l'état des bascules ou alors d'utiliser cette fonctionnalité uniquement lorsque le traitement est totalement terminé et que la reprise en état initial est possible.

Les alimentations internes du circuit se comportent comme un équivalent de résistances, capacités et inductances. Elles doivent être conçues pour que la transition

d'alimentation d'une partie du circuit ne propage pas de parasites trop importants qui risqueraient de compromettre la sûreté de fonctionnement du système [31].

1.1.3.3 Changement dynamique de tension et de fréquence

La tension d'alimentation joue un rôle important dans la consommation du circuit puisque son interaction dans l'équation de consommation est un terme au carré. Réduire la tension d'alimentation permet de réduire efficacement la puissance consommée. Cependant la vitesse de commutation des transistors est dépendante de la tension appliquée sur leur grille. Réduire la tension d'alimentation conduit à la diminution de la tension de grille et augmente les temps de propagation. La fréquence de fonctionnement ne peut plus être aussi élevée au risque de commettre des erreurs fonctionnelles.

La réduction de la tension d'alimentation s'accompagne alors par une réduction de la fréquence de fonctionnement, on parle de points de fonctionnements ou OPP (Operating Performance Points). Ces couples de tension-fréquence sont définis par les fabricants de circuits. Lors du passage d'un OPP à un autre de fréquence inférieure, les performances du système sont dégradées. Afin de conserver les performances optimales du système, le changement de point de fonctionnement doit être dynamique, c'est-à-dire pendant le fonctionnement. En fonction de la charge du système et selon les politiques d'énergie, l'OPP est adapté pour que le système fonctionne dans de bonnes conditions, notamment en respectant éventuellement des contraintes de temps. Cette technique, dénommée DVFS (pour *Dynamic Voltage and Frequency Scaling*), est couramment utilisée pour les processeurs [32]. Le choix de l'OPP à utiliser est réalisé par un module de gestion de la puissance en fonction de plusieurs critères dont celui de la performance (qualité de service) [33].

Les techniques DVFS ont un coût de conception lié à leur dynamicité, elles nécessitent une alimentation et un contrôleur adaptés. L'horloge est généralement générée par une PLL qui, comme les circuits d'alimentation lors du changement de tension, nécessite un temps de stabilisation. Par exemple, dans [34] les auteurs précisent qu'il faut 150 μs pour effectuer un changement de tension et verrouiller la nouvelle fréquence d'horloge sur une plateforme de réseau de capteurs basée sur un StrongARM SA-1100.

1.1.3.4 Accélérateurs matériels

L'accélération matérielle consiste à exécuter une fonction sur un circuit matériel dédié. Les accélérations obtenues sont généralement de plusieurs facteurs par rapport à une exécution sur processeur (exécution dite logicielle). De plus, ces circuits sont plus économes en énergie qu'une version logicielle pour l'exécution du même

calcul [35]. Ceux-ci sont adaptés à une opération spécifique, contrairement au processeur qui nécessite une gestion plus ou moins séquentielle d'instructions. De plus, les instructions processeur peuvent ne pas correspondre directement au calcul requis, plusieurs instructions sont alors nécessaires pour réaliser le même calcul, d'où une perte de performance et d'énergie.

En plus de leur cycle de développement très coûteux et complexe, le principal défaut des accélérateurs matériels est la non flexibilité. Après sa conception, un accélérateur n'est pas modifiable pour corriger une erreur, pour l'optimiser ou pour changer sa fonctionnalité à moins d'utiliser un circuit reconfigurable. Ceci peut aussi poser un problème de consommation puisque, à moins de pouvoir couper l'alimentation (power gating), l'accélérateur consomme de l'énergie même lorsqu'il n'est plus utilisé. Finalement, lorsque la diversité des fonctions ou algorithmes est importante, il faut autant de blocs accélérateurs matériels que de fonctions engendrant une hausse de la consommation statique et *idle*. Les circuits reconfigurables permettent d'apporter une réponse au manque de flexibilité de ces accélérateurs matériels et au partage des ressources matérielles.

1.2 Circuits logiques reconfigurables

Un circuit logique configurable est un circuit dont la fonction logique peut être configurée et est réalisée à partir d'éléments logiques de base. La notion de reconfigurabilité apparaît lorsqu'il est possible de modifier la configuration en cas de besoin.

Un circuit est dit reconfigurable dynamiquement s'il est possible de modifier sa configuration lorsqu'il est dans un système en cours de fonctionnement. Un circuit est dit reconfigurable partiellement ou reconfigurable partiellement et dynamiquement s'il est possible de modifier la configuration d'une zone de celui-ci, sans interférer sur les autres régions en cours de fonctionnement.

1.2.1 Architectures pour la reconfiguration

Plusieurs types d'architectures existent et permettent la reconfiguration. La reconfiguration peut s'appliquer à des opérateurs d'un processeur pour l'adapter au type d'application en cours d'exécution et améliorer les performances ou bien pour des régions plus grandes comme pour des accélérateurs matériels dans un SoC (*System-on-Chip* - Système sur puce), connectés à un ou plusieurs processeurs. Généralement, les données de configuration sont stockées dans des mémoires effaçables électriquement comme une EEPROM, une SRAM ou une FLASH ce qui permet de les modifier aisément.

Par la suite, nous allons traiter l'exemple des FPGAs car ceux-ci sont largement

répandus et permettent une grande flexibilité de configuration. Un circuit reconfigurable peut être représenté comme un circuit ayant deux couches, une couche composée d'une architecture dont les fonctionnalités (notamment opérations) sont configurables et une couche composée de la mémoire de configuration comme représenté Figure 1.6. Cette mémoire de configuration est la ressource qui stocke les informations de configuration et c'est dans cette mémoire que le *bitstream* est chargé pour appliquer une nouvelle configuration. Le *bitstream* est constitué des données de configuration brutes et est appelé ainsi puisqu'il s'agit d'un "flot de bits" qui est transmis directement à la mémoire de configuration via un contrôleur de reconfiguration[1].

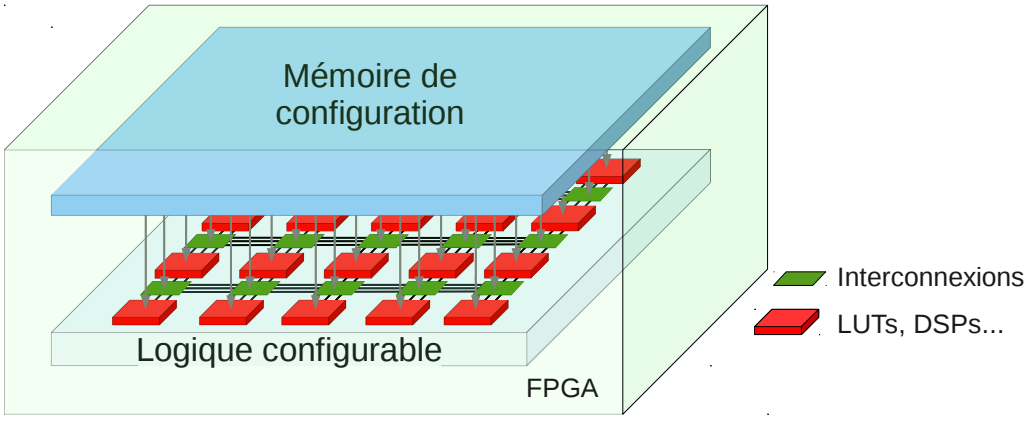

FIGURE 1.6: Représentation d'une architecture reconfigurable sur deux plans.

L'architecture des FPGAs est généralement découpée en éléments unitaires, appelés *slices* par exemple chez Xilinx, ces éléments étant composés de tables de vérités (*Look-Up Table* LUT) et de bascules. D'autres blocs spécifiques sont aussi présents, notamment les blocs d'entrées sorties et de gestion des horloges. De plus en plus de circuits reconfigurables incluent de la mémoire (BRAM) ou d'autres blocs spécifiques comme des DSPs (contenant des multiplieurs-accumulateurs performants). L'évolution actuelle des FPGAs tend vers des circuits hétérogènes et SoCs complets comme le Xilinx Zynq qui intégre un double ARM CortexA9 et un composant Virtex-7.

1.2.2 Méthodes de reconfiguration

La reconfiguration est l'opération qui consiste à charger les données de la nouvelle configuration dans la zone mémoire de configuration du circuit et à l'appliquer. La reconfiguration s'effectue selon trois méthodes principales ; la reconfiguration peut être complète, partielle ou différentielle.

[1]Cependant, les *bitstreams* peuvent être compressés, cryptés ou en partie interprétés.

1.2.2.1 Reconfiguration complète

La reconfiguration complète consiste à charger une nouvelle configuration pour l'ensemble du circuit. Celle-ci nécessite le transfert de la totalité des données depuis la mémoire de stockage externe vers la matrice de configuration. Pendant la reconfiguration, la configuration globale du circuit n'est pas valide, celui-ci est alors inopérant. La taille du *bitstream* pouvant être importante, le temps de reconfiguration et donc d'indisponibilité est non négligeable.

La reconfiguration complète nécessite un contrôleur externe pour gérer le transfert des données et sa prise en compte au niveau du système pour l'ajout de buffers, par exemple, pour sauvegarder les données transmises au circuit reconfiguré.

1.2.2.2 Reconfiguration dynamique partielle

La reconfiguration partielle consiste à uniquement modifier la configuration d'une région du circuit, la PRR (*Partial Reconfigurable Region*). Cette méthode permet de transférer une quantité plus petite de *bitstream*, ce qui réduit les temps de reconfiguration. De plus, si le circuit le supporte, la reconfiguration partielle n'empêche pas le bon fonctionnement des zones non concernées par le changement de configuration, le reste du circuit est opérationnel. On parle de reconfiguration dynamique partielle ou de reconfiguration dynamique. Cette technique a l'avantage de rendre transparente la reconfiguration tant que la région concernée ne retarde pas l'exécution de l'application.

Cette approche permet d'intégrer le contrôleur de la reconfiguration dans le circuit reconfiguré, celui-ci étant alors capable de se reconfigurer lui-même. Le FPGA peut se reconfigurer en fonction des besoins en accélérateurs matériels sans interrompre le système. Il n'y a pas besoin de contrôleur de reconfiguration externe et cette technique facilite l'utilisation d'un FPGA seul pour l'architecture du système. La reconfiguration dynamique partielle est supportée par les FPGAs de chez Xilinx, le XC6200 [36] et notamment la famille Virtex depuis le Virtex-II PRO [37].

1.2.2.3 Reconfiguration différentielle

La reconfiguration différentielle consiste à configurer uniquement les bits ou les mots, selon la granularité disponible, dont la configuration change. Ceci est une forme de compression qui permet, lorsque l'on connaît la configuration précédente de ne pas reconfigurer les ressources logiques réalisant les mêmes fonctionnalités entre deux configurations successives. L'avantage de cette méthode est de réduire la taille du *bitstream* et donc de réduire le temps d'indisponibilité de la région concernée. La reconfiguration différentielle peut être utilisée pour la reconfiguration dynamique

partielle pour améliorer les performances de cette dernière. De même que la reconfiguration dynamique partielle, elle est supportée par les composants de chez Xilinx.

1.2.3 Constructeurs et outils

Actuellement les deux constructeurs principaux de FPGA pour le calcul haute performance sont Xilinx et Altera. D'autres constructeurs comme Microsemi privilégient principalement les circuits reconfigurables à très faible consommation et la reconfiguration dynamique partielle n'est à notre connaissance pas encore supportée. La reconfiguration dynamique partielle étant gérée complètement depuis les Virtex-II pro en 2002, nous nous intéressons principalement aux FPGAs de Xilinx.

La suite d'outils proposée par Xilinx permet la conception complète d'un système composé de processeurs et d'une matrice reconfigurable destinée aux accélérateurs matériels. Les outils permettent la conception de systèmes utilisant la reconfiguration dynamique partielle. Cependant, cette technique requiert quelques précautions pour les interconnexions et un flot de conception légèrement différent avec un partitionnement obligatoire des zones susceptibles d'être reconfigurées dynamiquement. Malgré une amélioration constante de l'intégration de la reconfiguration dynamique dans les outils, l'impact de cette technique est difficile à évaluer sur un système complet. Il n'y a pas d'estimation de la consommation et la reconfiguration nécessite une gestion particulière de l'allocation des ressources pour assurer le bon fonctionnement d'une application.

1.3 Consommation des circuits reconfigurables

L'architecture d'un circuit configurable étant en général composée principalement d'interconnexions, de bascules et de fonctions logiques élémentaires, l'analyse et l'estimation de la consommation de ces circuits impliquent de prendre en compte l'utilisation de chacun de ces éléments. Cette section présente la répartition de la consommation dans les FPGAs ainsi que les méthodes pour réduire cette consommation.

1.3.1 Répartition de la consommation

Les éléments internes d'un FPGA sont en général les cellules logiques qui contiennent les LUTs et les bascules, les interconnexions, les blocs d'entrées/sorties, l'arbre d'horloge, les mémoires et les autres blocs spécifiques comme des DSPs, unités de calcul dédiées au traitement du signal et optimisées notamment pour calculer des MAC (*Multiply-ACcumulate* - multiplication puis accumulation).

L'analyse de la consommation globale passe par l'étude de la consommation de chaque élément, ceci dans le but de pouvoir estimer la consommation d'une fonction les utilisant et de la réduire. Dans [1], les auteurs effectuent l'analyse de consommation sur deux circuits, le premier de chez Altera (Flex10k100) et le second de chez Xilinx (XC4003). Les vecteurs de tests sont adaptés pour mettre en évidence la consommation distincte des interconnexions, des LUTs, des entrées sorties, de l'arbre d'horloge, des bascules ou de la mémoire. Les résultats sont présentés Figure 1.7 et montrent que la proportion de consommation des éléments des deux FPGAs est semblable. Environ 50% de la consommation est liée aux cellules logiques, 34% de la puissance est requise par les interconnexions, puis 10% pour l'arbre d'horloge.

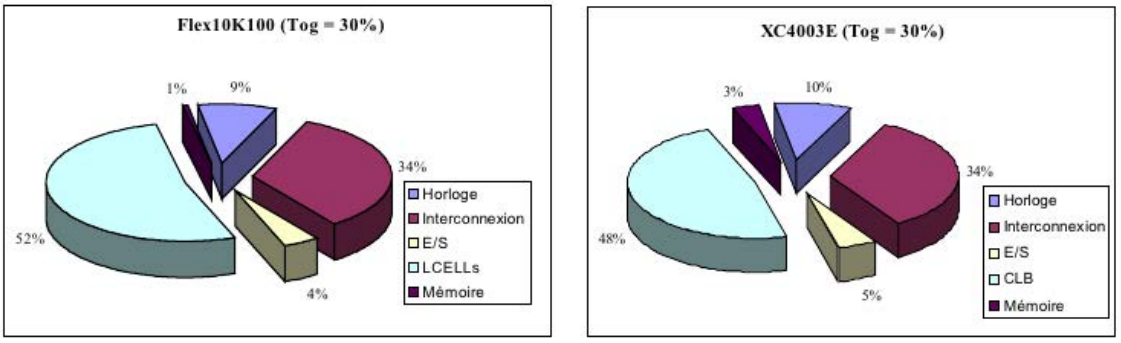

FIGURE 1.7: Distribution de puissance dans deux FPGAs. [1]

Ces résultats montrent, même si les technologies ont évolué, que les algorithmes de placement et de routage doivent optimiser l'utilisation des cellules logiques au maximum et doivent réduire la longueur et le nombre des interconnexions pour réduire la consommation d'une application sur un FPGA.

1.3.2 Estimation de la consommation

La section précédente a montré que la consommation d'un FPGA est liée à l'utilisation de ses ressources principales. L'estimation de la consommation dans ce type de circuit doit alors prendre en compte les ressources utilisées pour avoir une bonne précision d'estimation.

Les méthodes générales d'estimation présentées dans le paragraphe 1.1.1 sont bien entendu applicables pour estimer la consommation des circuits configurables. Ainsi la première méthode d'estimation bas niveau se fait au minimum après la synthèse logique, lorsque des informations sur l'utilisation des ressources sont disponibles, ou mieux après le placement et routage, ce qui permet de connaître la longueur des connexions. [38] propose une caractérisation de la consommation des blocs élémentaires d'un FPGA. La quantité des ressources utilisées et la longueur des intercon-

nexions sont obtenues des informations de placement et routage. L'estimation de la consommation obtenue à partir de cette caractérisation a une erreur de 18%. Xilinx propose un outil dénommé Xilinx Power Estimator [39] qui permet, également après récupération des informations de synthèse, d'estimer la consommation statique et dynamique pour une tension, température et un taux d'activité donné.

L'estimation à plus haut niveau est moins précise et dépend beaucoup de la capacité des outils de conception à optimiser l'utilisation des ressources. [27] propose une méthodologie permettant l'estimation de la consommation d'un système reconfigurable à haut niveau d'abstraction grâce à des modèles de consommation à plusieurs niveaux : système, architectural et algorithmique. Cette approche permet de prendre en compte la consommation énergétique au plus haut niveau de conception. Puis en fonction des paramètres qui s'affinent au fil de la description et de la conception, le niveau des modèles d'estimation peut être adapté et la précision de l'estimation est améliorée. Les modèles de consommation sont initialement construits à partir d'un découpage du système en blocs, tel que précisé plus tôt dans ce chapitre. La consommation est analysée en fonction des paramètres fonctionnels de ce bloc et l'équation d'estimation est construite.

Il existe plusieurs outils et techniques d'estimation de la puissance consommée par un circuit reconfigurable, cependant ces techniques n'incluent pas le coût de la reconfiguration. L'impact de la reconfiguration est en effet difficile à appréhender et cette partie est détaillée par la suite.

1.3.3 Techniques de réduction de la consommation dans les architectures reconfigurables

Avec l'apparition de la reconfiguration, notamment dynamique, de nouvelles techniques ayant pour but de réduire la consommation ont vu le jour. La reconfiguration dynamique partielle permet par exemple de modifier le routage du signal d'horloge dans le but de mettre en place du *clock gating* ou d'opérer des variations de fréquence, ceci constitue le premier point de cette section. La seconde partie présente une autre utilisation plus répandue de la reconfiguration dynamique, qui consiste à ordonnancer et effectuer des choix de configuration des tâches matérielles. Finalement, la reconfiguration dynamique permet la réutilisation des ressources et l'implémentation d'accélérateurs exploitant le parallélisme des tâches, pour une meilleure efficacité énergétique, est envisageable lorsque la surface est disponible.

1.3.3.1 Optimisation dynamique de l'arbre d'horloge

La reconfiguration dynamique permet de modifier certains paramètres de routage de l'horloge. Il est possible d'adapter le routage de l'horloge pour arrêter sa pro-

pagation sur les bascules et ainsi diminuer la consommation dynamique. Ceci peut être effectué par une reconfiguration dynamique partielle de la partie liée à l'horloge d'une PRR et réduire sa consommation lorsque la tâche occupant la PRR n'est plus utilisée. En ne modifiant que la distribution de l'horloge, selon le principe de la reconfiguration différentielle, les temps de reconfiguration sont beaucoup plus courts que la reconfiguration complète de la PRR. Ainsi [40] propose une méthode de *clock gating* en utilisant la reconfiguration dynamique sur un Xilinx Virtex-4 et un Virtex-5. Les auteurs indiquent qu'il y a trois endroits où l'arbre de distribution de l'horloge peut être interrompu et où le *clock gating* peut avoir lieu, en fonction de la granularité souhaitée. La distribution de l'arbre d'horloge peut être contrôlée :

- au niveau de la répartition en domaines d'horloge,
- au niveau de chaque colonne de CLBs,
- au niveau de chaque bascule.

Cette méthode de *clock gating* par la reconfiguration apporte de meilleurs résultats énergétiques que l'implémentation du *clock gating* au niveau RTL avec la logique programmable. En effet, la reconfiguration agissant sur des circuits de gestion de l'horloge déjà présents, il n'y a pas d'augmentation du chemin critique, ni de la surface, ni détérioration de la précision du signal d'horloge. Cependant l'énergie pour effectuer la reconfiguration est à prendre en compte. Un gain énergétique de 62% en moyenne est obtenu sur des exemples d'implémentation de FIFO. Dans [41], les auteurs proposent d'optimiser l'arbre d'horloge lors de la phase de placement en groupant les bascules qui sont connectées sur l'horloge principale sur une même colonne de ressources CLBs. Ceci permet de désactiver l'arbre d'horloge et de réduire l'étendue de celui-ci sur l'ensemble du FPGA. La complexité et la capacité de l'arbre d'horloge sont réduites ce qui a pour effet direct de diminuer la consommation dynamique. Les auteurs obtiennent une réduction de la consommation de 13.5% en moyenne grâce à l'utilisation de cette technique sur un ensemble d'applications.

La reconfiguration dynamique permet de modifier, en ligne, la fréquence d'horloge en modifiant directement les paramètres du bloc gestionnaire de l'horloge sans avoir à piloter ce bloc par une interface propre [42]. La surface requise est réduite dans le cas où le contrôleur de reconfiguration est déjà implémenté, il n'y a pas de mécanismes supplémentaires à implémenter pour la gestion des fréquences d'horloge. Dans cet exemple, un traitement est effectué pendant 7 ms, la reconfiguration dure 0.32 ms et la période totale est de 100 ms. L'expérience est effectuée sur un Spartan-3 (expérience théorique car ce Spartan ne supporte pas la reconfiguration dynamique) et la réduction de consommation estimée est de 8%.

1.3.3.2 Ordonnancement et allocation des ressources

La reconfiguration dynamique amène la problématique de la gestion de l'implémentation des tâches. La reconfiguration dynamique doit alors être supervisée par un ordonnanceur des tâches de manière semblable à des architectures multi-processeurs mais elle nécessite tout de même une gestion spécifique pour permettre un fonctionnement correct [43]. Cette gestion est généralement effectuée par un ordonnanceur de tâches, comme proposé dans [44], qui doit décider, à la volée de l'exécution, l'exécution spatio-temporelle des tâches. De plus, il doit être possible de faire de la préemption de tâches avec sauvegarde et restauration de contexte [45] et ré-allocation. C'est-à-dire qu'une tâche configurée et en cours d'exécution peut être suspendue et sauvegardée pour permettre l'exécution d'une autre tâche à priorité plus élevée, puis la tâche interrompue peut être reconfigurée sur une région disponible et reprendre le cours de son exécution. L'instant de la reconfiguration est important, il faut éviter qu'une tâche configurée ne soit laissée longtemps inutilisée avant son exécution à cause de sa consommation statique et dynamique de repos (consommation *idle*) [46].

1.3.3.3 Effacement de la configuration

Une autre approche consiste à utiliser la reconfiguration dynamique partielle pour effacer la configuration d'une zone non utilisée à l'aide d'une configuration spécifique, dite *blank*, qui permet de réduire la consommation. La configuration *blank* permet de déconnecter les interconnexions et effacer la configuration des LUTs, mais elle permet aussi de désactiver la propagation de l'horloge vers les bascules et de désactiver des ressources spécifiques comme par exemple les BRAMs. Dans [47], les auteurs comparent cette méthode avec les techniques classiques de *clock gating*. L'étude montre que dans certaines conditions, cette méthode permet de réduire la consommation et même d'obtenir un meilleur gain énergétique que la technique de *clock gating*. Les conditions tiennent compte de la performance (vitesse et énergie) de la reconfiguration et du temps de maintient de l'accélérateur en configuration *blank*. C'est-à-dire qu'il faut que l'énergie récupérée grâce à l'effacement soit supérieure à l'énergie consommée pour appliquer la reconfiguration. Ceci implique, pour un ordonnanceur en ligne, la nécessité de connaître à l'avance, ou d'estimer, quand cette ressource sera à nouveau utilisée afin de déterminer si l'effacement de la configuration permet d'économiser de l'énergie ou non. Cette information n'est pas toujours facile à obtenir, d'autant plus si le système est susceptible de configurer d'autres accélérateurs sur la même région. Pour cette raison, il y a un réel besoin d'outils d'exploration afin de dimensionner les ressources matérielles et d'appréhender l'impact de l'usage de la reconfiguration dynamique sur l'efficacité énergétique du système. L'ordonnan-

cement des tâches matérielles permet d'optimiser l'utilisation des ressources et de réduire la surface nécessaire à l'exécution d'une même application, cependant il est aujourd'hui difficile d'appréhender l'impact de l'utilisation d'accélérateurs reconfigurables dynamiquement sur la consommation globale d'un système. Ce point est une problématique pour l'utilisation de la reconfiguration dynamique partielle dans les systèmes embarqués. De plus, la liberté d'implémentation offerte par les accélérateurs matériels, telle que l'exploitation du parallélisme, rend les choix de conception encore un peu plus complexes.

1.3.3.4 Exploitation du parallélisme

L'utilisation du parallélisme prend tout son sens sur des architectures reconfigurables puisqu'il est possible de déterminer au moment de la conception le nombre d'opérateurs à implémenter. Le parallélisme permet de diminuer le temps d'exécution d'un algorithme en exécutant des opérations en parallèle lorsque les dépendances d'exécution et de données le permettent. Une possibilité pour exploiter le parallélisme d'un algorithme est d'utiliser les transformations de code [48]. Le déroulage de boucles fait partie de ces transformations de code utilisées en synthèse de haut niveau pour bénéficier du parallélisme [49]. Les outils les plus connus sont *ImpulseC* [50], *CatapultC* [51] et plus récemment Vivado HLS [52] permettent l'exploration du parallélisme dans les algorithmes. Des travaux ont été menés sur l'étude des transformations de code pour la réduction de la puissance, de l'énergie, de la surface ou encore du temps d'exécution d'un algorithme [53]. La variation du niveau de parallélisme permet une réduction du temps d'exécution mais affecte aussi la complexité de gestion des boucles, des opérateurs et des ressources (mémoires principalement), ce qui risque de réduire la fréquence de fonctionnement de l'algorithme et compromettre ses performances. Dans [54], un niveau de parallélisme "optimal" est obtenu à partir d'une fonction tenant compte des estimations du coût de la complexité du contrôleur de boucle et des gains obtenus grâce au parallélisme. Ce niveau est sélectionné pour ne pas compromettre les performances de l'accélérateur, à cause de l'augmentation de la latence, en conservant la fréquence de fonctionnement choisie.

Cependant, si l'augmentation du niveau de parallélisme permet une accélération dans une certaine mesure, la surface de l'accélérateur augmente et la reconfiguration de cet accélérateur risque de coûter plus cher. Ceci amène un nouvel obstacle pour l'implémentation matérielle dynamique dans les systèmes à contraintes de temps ou d'énergie fortes.

1.4 Coût de la reconfiguration sur le système

La reconfiguration dynamique permet la mise en place et l'utilisation de nouveaux mécanismes de réduction de la consommation énergétique dans les circuits reconfigurables, cependant celle-ci a des inconvénients. La reconfiguration dynamique a un impact sur la surface, les performances et, finalement, l'énergie.

1.4.1 Impact sur la surface

La reconfiguration dynamique autorise l'augmentation du taux d'utilisation des ressources du FPGA et ceci permet de réduire la surface nécessaire à l'exécution d'une application. Par exemple, [55] montre que la reconfiguration dynamique permet d'augmenter la densité d'un réseau de neurones de 500%. Néanmoins, il existe un surcoût de surface puisqu'il est nécessaire d'implémenter un contrôleur de reconfiguration ainsi qu'une mémoire de stockage des *bitstreams*, en général sur une mémoire externe. Il est à noter qu'il est souvent nécessaire de positionner des *buffers* entre les zones reconfigurables et la partie statique du FPGA. Ceci a pour but de ne pas répercuter les variations erronées des signaux lors de la reconfiguration, notamment lorsqu'il s'agit d'un bus de communication. Le coût en surface de ces *buffers* dépend naturellement du nombre d'entrées/sorties à protéger mais cette surface reste faible. Ces *buffers* sont constitués d'une bascule ou d'une LUT par signal.

La reconfiguration dynamique, telle qu'elle est implémentée actuellement nécessite un partitionnement du FPGA en PRR (*Partial Reconfigurable Region*) qui sont des zones reconfigurables dynamiquement. Cependant, le découpage en régions reconfigurables peut accroître la quantité de ressources non utilisées. Le partitionnement doit être réalisé suivant le grain minimum de reconfiguration et la taille de la PRR est un arrondi supérieur aux besoins des tâches. De plus toutes les tâches n'ont pas les mêmes besoins en ressources, et de petites tâches peuvent être implémentées sur de larges régions ce qui augmente d'autant plus la quantité de ressources inutilisées. Dans [56], les auteurs proposent un outil de partitionnement des ressources matérielles et le taux de ressources inutilisées est de 38%. Ce partitionnement, statique, doit être effectué manuellement avant l'implémentation des tâches. Il est alors difficile d'optimiser l'utilisation des ressources et de déterminer la taille du FPGA nécessaire. Dans [57], les auteurs présentent une méthode basée sur une modélisation à haut niveau permettant d'optimiser l'espace et la flexibilité du composant reconfigurable en fonction des tâches à exécuter, grâce à la mise en place d'algorithmes d'ordonnancements des tâches matérielles. Dans l'application de traitement vidéo donné en exemple, cinq régions reconfigurables sont utilisées pour exécuter dix tâches. La technique d'ordonnancement proposée permet un taux d'utilisation des

ressources de plus de 90%.

La reconfiguration dynamique peut réduire la surface nécessaire à l'exécution d'une même application, mais cette capacité doit être explorée méthodiquement ; le choix des zones reconfigurables et l'implémentation des accélérateurs matériels sont un problème complexe à eux seuls. Ce problème est d'autant plus complexe que la reconfiguration dynamique a un coût temporel et peut impacter les performances du système.

1.4.2 Impact sur les performances

La reconfiguration dynamique consiste à transférer les nouvelles données de configuration vers la matrice de configuration, ceci nécessite un certain temps qui dépend des performances du contrôleur de reconfiguration. Ainsi l'accélération de la reconfiguration dynamique est un enjeu important pour permettre son utilisation dans des systèmes ayant des contraintes de temps sévères. [58] propose une estimation des temps de reconfiguration en tenant compte de l'ensemble des composantes nécessaires, de la mémoire au contrôleur de reconfiguration, en passant par les bus et la gestion du processeur.

Le contrôleur de reconfiguration proposé par Xilinx, appelé *xps_hwicap* [59], est dépendant d'un processeur, soit un *MicroBlaze* [60] ou un PowerPC [61], pour effectuer une reconfiguration. Grâce à la grande capacité de stockage de la mémoire non volatile comme la CompactFlash, il peut contrôler de gros *bitstreams* sans faire appel à la compression. Cependant, il dispose d'un débit de reconfiguration très faible. Dans nos expériences, le débit enregistré est de 180 KB/s pour ce contrôleur. Dans [62], les auteurs ont mesuré les performances de *xps_hwicap* en utilisant la mémoire cache du processeur et ont atteint 14.5 MB/s de débit effectif de reconfiguration. Ce contrôleur, à cause de son interaction avec le processeur, ne permet une reconfiguration qu'à très bas débit. Pour cette raison, d'autres contrôleurs ont été développés et permettent d'obtenir des performances plus élevées :

- $BRAM\_HWICAP$ [62] met en place un DMA connecté sur un BRAM qui permet d'atteindre la vitesse maximale garantie de ICAP (port interne permettant la reconfiguration dynamique partielle dans les FPGAs Xilinx Virtex), 400 MB/s à 100MHz. Cependant, ce contrôleur est connecté sur le bus du processeur pour permettre le chargement du BRAM et donc sa fréquence est limitée à 120MHz. Ceci augmente le temps de reconfiguration puisqu'il faut charger le BRAM puis effectuer la reconfiguration à partir du BRAM. De plus, l'ensemble du *bitstream* doit être sauvegardé dans le BRAM, un BRAM qui ne contient pas plus de 36 kb. Même s'il est possible de regrouper les BRAMs pour

augmenter la capacité de stockage, ce sont des ressources limitées du FPGA. Ceci limite la taille des *bitstreams* supportés par $BRAM\_HWICAP$.

- MST_ICAP [62] permet un stockage de *bitstreams* de plus grande taille en utilisant une mémoire DDR2 SDRAM externe. Cependant, la vitesse de transfert de la DDR est plus faible que pour un BRAM (qui est interne au FPGA, c'est une mémoire rapide dont les longueurs des interconnexions sont réduites) et limite la vitesse de reconfiguration.

- $FlashCAP_{(i)}$ [63] utilise un algorithme de compression, X-MatchPRO [64], qui a un taux de compression important sur les *bitstreams* (74.2%). Mais la fréquence maximale est toujours limitée à 120 MHz et la fréquence de reconfiguration effective est encore plus faible et ne permet qu'un débit de reconfiguration de 385 MB/s.

- $FaRM$ [65, 66] inclut l'ensemble des caractéristiques innovantes des autres contrôleurs de reconfiguration, c'est-à-dire, un DMA avec une DDR externe pour accélérer les transferts de cette mémoire accompagnée d'un *buffer* en BRAM pour éviter les défauts d'approvisionnement en données. $FaRM$ supporte la compression du *bitstream*. La décompression est effectuée en ligne ce qui permet d'augmenter les tailles de *bitstream* sans augmenter le stockage. La compression permet d'augmenter le débit effectif de reconfiguration grâce à une réduction des transferts depuis la mémoire externe. $FaRM$ permet une reconfiguration à une fréquence maximale élevée de 200 MHz, soit 800 MB/s.

Le tableau 1.1 montre les comparaisons de performances, débit de reconfiguration et fréquence de fonctionnement maximale, entre différents contrôleurs de reconfiguration.

Table 1.1: Comparaison de différents contrôleurs de reconfiguration

Contrôleur de Reconfiguration	Bande passante (MB/s)	Freq. max (MHz)
xps_hwicap[59]	14.5	120
MST_ICAP[62]	235	120
FlashCAP$_i$[63]	358	120
BRAM_HWICAP[62]	371	120
FaRM[65]	800	200

Lors de la reconfiguration, la région concernée est indisponible pendant toute la procédure. Hormis des effets de bord (adressage de la DDR par exemple), on

peut considérer que la reconfiguration se déroule à vitesse fixe et que le temps de reconfiguration est proportionnel à la taille du *bitstream* (sauf lors de l'utilisation de la compression qui peut engendrer des débits variables).

Grâce à l'amélioration des performances des contrôleurs de reconfiguration, l'impact temporel de la reconfiguration est réduit mais il est toujours présent. Lors de la reconfiguration, la puissance statique est toujours présente ainsi que la consommation des autres parties du FPGA, le temps de reconfiguration engendre une perte énergétique.[67] montre que la vitesse de reconfiguration impacte beaucoup l'efficacité énergétique d'un système reconfigurable et que la conception d'un contrôleur de reconfiguration performant est très importante. Leur étude met en valeur la consommation énergétique de la gestion de la reconfiguration dynamique puisque l'énergie consommée varie de 19% entre un contrôleur de reconfiguration interne au FPGA et un contrôleur externe dont la consommation n'est pas comptabilisée. La procédure de reconfiguration a également un impact sur la puissance du système.

1.4.3 Impact sur la puissance

La reconfiguration est, de part sa nature, un transfert de données (de configuration) depuis une mémoire, généralement externe, vers la mémoire de configuration. Cette opération génère de l'activité et induit une consommation dynamique. La puissance consommée par la reconfiguration est rarement évoquée, souvent la littérature évoque une consommation d'énergie, évidemment liée aux coûts temporels de la reconfiguration, mais la surconsommation en puissance engendrée par l'activité de reconfiguration et sa caractérisation ne sont pas détaillées. Cependant, [68] note une consommation plus importante du cœur du FPGA durant la reconfiguration d'un Virtex II, jusqu'à 100mW et pointe la difficulté d'estimer la puissance consommée par une application reconfigurée. La thèse [69] évoque la consommation de la reconfiguration. La puissance mesurée est constante au cours de la reconfiguration dynamique partielle sur un FPGA Xilinx Virtex-II pro. Dans [70], les auteurs évoquent la présence de courts-circuits au niveau des interconnexions lors du changement de la configuration sur un FPGA ATMEL, ce qui provoque des surconsommations au cours de la reconfiguration. Il n'y a pas de modèle de consommation tenant compte de ces variations.

[67] montre qu'une réduction de la surface de 87% du FPGA (dans le cas étudié, un filtre FIR) est possible grâce à l'augmentation du taux d'utilisation des ressources, apportée par la reconfiguration dynamique. Grâce à cette réduction de la surface, la consommation statique et *idle* du système peut être réduite. La reconfiguration dynamique partielle peut alors être utilisée dans des applications où la puissance consommée et la performance sont des considérations importantes [71]. Cependant,

le manque de caractérisation/modèles et le manque d'outils pour estimer l'impact de la mise en application de cette technique rendent très difficile l'évaluation du gain énergétique potentiel pour une application lors de la conception d'un système.

1.5 Modélisation à haut niveau et estimation de la consommation

L'impact de la puissance et l'énergie étant de plus en plus important dans les systèmes embarqués, la prise en compte de ces paramètres au plus tôt dans les phases de conception est primordiale. Pour cette raison, l'estimation de la consommation est généralement effectuée à haut niveau (niveau système). La modélisation est une phase importante qui permet, par la suite, de vérifier plusieurs caractéristiques dont la consommation et les performances d'un système complet.

1.5.1 Approche MDE

La modélisation a pour objectif de mieux maîtriser la complexité de conception des systèmes et notamment des systèmes sur puce dont la complexité ne cesse de croître. Issue du domaine des systèmes informatiques, la méthodologie MDE (*Model Driven Engineering*) est une approche de modélisation qui propose langages, outils et concepts pour manipuler les modèles et élever le niveau d'abstraction dans le développement de logiciels. L'approche MDE encourage l'utilisation de plusieurs niveaux d'abstraction. Les transitions entre les niveaux peuvent être automatisées grâce aux outils de transformation de modèles et à la génération de code.

La modélisation suivant la méthode MDE permet progressivement de :

- donner une vision fonctionnelle du système et de son environnement, selon le modèle CIM (*Computation Independent Model*) ;
- spécifier le système indépendamment de la plateforme, selon le modèle PIM (*Platform Independent Model*) ;
- décrire la plateforme matérielle, selon le modèle PDM (*Platform Dependent Model*) ;
- affecter le système sur sa plateforme, selon le modèle PSM (*Platform Specification Model*).

Plusieurs langages et concepts sont proposés pour la modélisation suivant la méthodologie MDE, parmi ceux-ci notons UML, SYSML, MARTE et AADL.

UML [72] est un langage de modélisation graphique destiné à la modélisation des logiciels. L'atout principal de UML est son très haut niveau d'abstraction et les multiples vues en fonction du niveau de la conception. La version UML2.0 apporte de nombreux formalismes permettant la modélisation des systèmes embarqués et matériels. Malgré la similitude avec la conception des systèmes matériels, UML n'est pas adapté à la description comportementale ou structurelle. Pour cela des extensions à UML ont été proposées comme UML-SystemC [73], ce qui permet d'associer à UML les informations nécessaires pour passer à la description du système en SystemC, un langage de description matérielle.

SysML [74] est un langage de modélisation basé sur UML auquel est ajouté le formalisme nécessaire pour la conception des systèmes embarqués. SysML permet de décrire, analyser et vérifier la modélisation grâce à l'introduction de nouveaux paramètres incluant les contraintes de conception. SysML permet une approche moins conceptuelle que UML vis à vis de la formalisation logicielle.

MARTE [75] est aussi basé sur UML et permet une approche des systèmes complexes hétérogènes et multiprocesseurs y compris l'analyse et la vérification. Les extensions apportées à MARTE permettent maintenant de considérer la reconfiguration dynamique [76] ce qui lui donne un avantage par rapport aux autres langages. Les aspects temps réel sont supportés [77] ce qui permet d'explorer, avec les techniques d'analyse de l'ordonnancement, les implémentations qui satisfassent les contraintes de temps.

AADL, pour *Architecture Analysis & Design Language*, initialement dédié à l'avionique (*Avionics Architecture Description Language*), est un langage de description d'architecture. Il est utilisé pour modéliser les architectures matérielles et logicielles notamment pour les systèmes embarqués et les systèmes à contraintes temps réel [78] [79]. AADL est un langage textuel mais des outils permettent d'obtenir des représentations graphiques en arbre ou complètement graphiques. La modélisation permet de décrire la structure d'un système comme étant un assemblage de composants logiciels et matériels et permet de structurer les composants en sous-composants ayant des liens entre eux, flot de contrôle ou flot de données.

1.5.2 Plateforme OpenPEOPLE

La plateforme du projet OpenPEOPLE, dans lequel s'inscrit ce travail [80], porte en particulier sur la modélisation en consommation (énergie, puissance) pour des systèmes hétérogènes complexes. Une approche du type MDE à plusieurs niveaux d'abstraction est alors logiquement employée. L'objectif de cette plateforme est de proposer : (i) une modélisation des systèmes hétérogènes dans le but de pouvoir estimer plusieurs paramètres, notamment la consommation, (ii) des optimisations de la

consommation et des performances, (iii) des mesures physiques sur carte pour valider les estimations ou construire les modèles inexistants. La phase d'estimation et d'optimisation vise une exploration sur plusieurs architectures pour un même système. L'approche MDE convient dans ce cas puisque la modélisation permet une séparation du système, de sa description, de la plateforme et de l'implémentation/allocation (*mapping*). AADL est un langage de modélisation qui respecte l'approche MDE et a été choisi pour être le langage de modélisation de la plateforme OpenPEOPLE. La modélisation AADL se fait actuellement dans un environnement Eclipse qui permet une vue sous forme d'arbre ou une vue graphique de la description. Il existe trois outils principaux pour la manipulation de la modélisation AADL sous forme de plugin pour Eclipse. OSATE permet de manipuler et éditer les fichiers AADL et comporte plusieurs outils d'analyse, notamment pour l'analyse d'ordonnancement, d'allocation et de vérifications des connexions. ADELE permet de générer une représentation graphique de la modélisation AADL et permet de contrôler visuellement la description du système. RDAL (langage de vérification d'exigences) permet d'exprimer des contraintes puis de les vérifier lors de la modélisation du système sur la plateforme OpenPEOPLE. Néanmoins, AADL à l'origine principalement destiné à la modélisation logicielle est adapté pour pouvoir considérer les circuits hétérogènes et notamment la reconfiguration dynamique, point qui est abordé plus en détail dans le chapitre 4.

1.6 Conclusion et problématique

L'état de l'art est riche en modèles de consommation et méthodes pour réduire la consommation énergétique des systèmes embarqués. Grâce aux modèles d'estimation de la consommation à différents niveaux, il est possible de vérifier l'efficacité des méthodes de réduction de la consommation énergétique avant de déployer un système. Les outils de modélisation à haut niveau sont souvent développés et performants pour les logiciels et l'exécution sur processeur, or les circuits embarqués sont de plus en plus complexes et embarquent maintenant une (ou plusieurs) ressource(s) reconfigurable(s). L'intégration d'une ressource reconfigurable est plus complexe que l'emploi d'un accélérateur matériel dédié. En effet, l'architecture reconfigurable est plus flexible, sa configuration peut être modifiée en cours de fonctionnement. La reconfiguration dynamique introduit des surcoûts en temps et en énergie et ceci a un impact sur les performances des systèmes embarqués. L'état de l'art révèle notamment que si des modèles de délais de reconfiguration existent, les modèles de consommation ne sont pas clairement exposés. Il est possible d'évaluer ou de mesurer l'impact de la reconfiguration dans un scénario précis et restreint, la reconfiguration

apporte des possibilités de réduction de la surface et de la consommation énergétique. Cependant, les outils actuels ne permettent pas d'explorer les différentes possibilités d'implémentation des tâches et d'allocation des ressources logicielles et matérielles avec pour objectif l'estimation et la réduction de la consommation d'énergie pour un système complet.

Les prochains chapitres proposent une contribution pour répondre à cette problématique, tout d'abord en présentant une étude de l'impact du niveau de parallélisme sur la consommation énergétique et le temps d'exécution sur une architecture reconfigurable. Le chapitre suivant établit une modélisation de la consommation de la reconfiguration dynamique partielle. Finalement, le dernier point concerne la modélisation haut niveau et l'exploration des possibilités d'implémentation en vue de proposer des réductions de la consommation énergétique.

2 Étude de l'impact du parallélisme

Contenu

2.1 Introduction

Les capacités de traitement des FPGAs ne sont plus à démontrer et permettent d'atteindre des niveaux de performance bien supérieurs à des exécutions séquentielles sur un processeur avec une meilleure efficacité énergétique. La prise en charge du concept de reconfiguration dynamique apporte une souplesse de plus en plus souvent nécessaire pour supporter l'exécution des applications actuelles sur les FPGAs. Ces caractéristiques ont naturellement conduit les concepteurs de SoCs à intégrer ce type de ressources au sein de leur *design*. La reconfiguration dynamique permet une exécution séquentielle des tâches matérielles sur une même région. Cette fonctionnalité nécessite une gestion particulière des accélérateurs matériels qui peut être supportée par un système d'exploitation (OS - *Operating System*). L'OS peut effectuer cette gestion en ligne en fonction des contraintes extérieures et temps réel de l'application. Afin d'offrir plus de flexibilité à l'OS pour optimiser l'exécution globale d'une application, il peut être intéressant de définir plusieurs implémentations pour chaque (ou certaines) tâche(s). Obtenir plusieurs compromis performance/énergie/surface pour une même tâche permet à l'OS d'effectuer des choix d'implémentation pour respecter les contraintes au cours de l'exécution tout en réduisant la consommation en puissance ou en énergie. L'exploitation du parallélisme d'une tâche (si elle s'y prête) permet d'influencer ce compromis en proposant des versions de chaque tâche avec plusieurs niveaux de parallélisme. La description d'une tâche en utilisant le parallélisme ne change pas le nombre d'opérations à exécuter par rapport à la description

séquentielle de cette tâche. Avec ce type de transformation, la consommation en énergie d'une tâche spécifique peut cependant être améliorée grâce à une meilleure utilisation des ressources disponibles et grâce aux gains de performance obtenus qui permettent la diminution de la fréquence d'horloge [81].

Ce chapitre présente les résultats qui montrent que l'utilisation du déroulage de boucle en synthèse de haut niveau pour la génération de blocs accélérateurs matériels permet de varier les performances temporelles et énergétiques. Les choix d'implémentation de ces différentes solutions ont un impact sur les performances du système et ces choix peuvent être effectués en fonction des contraintes de temps réel, d'énergie ou de surface disponible. Ces résultats sont importants pour le système d'exploitation qui gère l'exécution et l'allocation des tâches, grâce à la reconfiguration dynamique partielle, pour satisfaire des contraintes de performance, d'énergie et de taille des régions reconfigurables. L'exploitation du parallélisme engendre une augmentation de la surface de l'accélérateur matériel. L'augmentation de la surface accroît la quantité de données de configuration nécessaires pour cet accélérateur et le coût de la reconfiguration dynamique risque d'être plus important. La dernière section de ce chapitre propose une formalisation de la consommation d'énergie lors de l'utilisation de la reconfiguration dynamique partielle dans une application afin d'apprécier l'intérêt de cette technique.

2.2 Procédure expérimentale

2.2.1 Plateforme

La plateforme ML550 proposée par Xilinx est utilisée pour nos expériences. Cette carte inclut un FPGA Virtex-5 (XC5VLX50T). L'alimentation de la carte est découpée en 5 rails qui alimentent le cœur du FPGA, les entrées sorties et les composants externes. Chaque rail possède une résistance de mesure du courant (résistance Kelvin) pour obtenir l'intensité circulant notamment dans le cœur et les entrées/sorties du FPGA et d'en déduire la puissance consommée.

La résistance de mesure du courant est de très faible valeur pour limiter les chutes de tension notamment pour le cœur du FPGA où des variations de quelques dizaines de millivolts pourraient compromettre sa stabilité. La résistance des rails d'alimentation est de 10 $m\Omega$ ce qui permet de limiter la chute de tension à 10 mV dans le cas où le cœur du FPGA consomme 1 W.

La tension mesurée aux bornes de la résistance est donc de très faible amplitude et elle est problématique pour mesurer des variations de consommation de quelques milliwatts. Une mesure précise peut se faire avec un voltmètre de précision sauf que le voltmètre ne mesure que la tension moyenne et les variations inférieures à la

seconde ne peuvent être interprétées.

L'utilisation d'un oscilloscope s'impose pour analyser précisément les variations du courant. Cependant les oscilloscopes ont une précision plus faible qu'un multimètre. Le calibre minimum de l'oscilloscope utilisé, le Agilent DSO7034B, est de 2 mV/div soit l'équivalent de 200 mA/div avec la résistance disponible sur la carte. Cette précision est trop faible. De plus la résistance étant placée sur le rail d'alimentation, il faut utiliser une sonde différentielle.

Pour remédier à ces problèmes, une carte avec un amplificateur de haute précision est conçue. L'amplificateur possède un gain de 100 et permet de convertir la tension aux bornes de la résistance en une tension directement analysable par l'oscilloscope et directement interprétable. La tension mesurée en sortie d'amplificateur est égale au courant circulant dans le rail d'alimentation et permet de visualiser le courant avec une précision de 2 mA/div avec l'oscilloscope. Le schéma de mesure est présenté sur la Figure 2.1.

L'amplificateur utilisé, le INA333 de Texas Instruments, a une bande passante de 3.5 kHz pour un gain de 100. La précision de l'amplificateur est de $\pm 0.25\%$. Dans notre cas, nous nous intéressons principalement aux courants et à la puissance consommés par le cœur du FPGA.

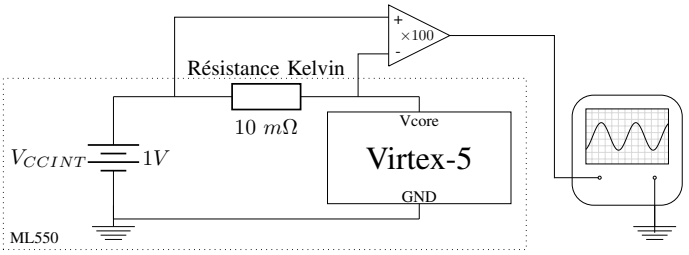

FIGURE 2.1: Montage amplificateur pour la mesure et l'analyse de la consommation sur FPGA.

2.2.2 Multiplication matricielle

Il est fréquent de mettre au point les blocs accélérateurs matériel pour obtenir le bon compromis entre la fréquence de fonctionnement, la surface et la consommation en puissance. Beaucoup d'algorithmes portés sur des accélérateurs matériels exécutent des opérations qui peuvent être parallélisées. Il est alors possible de choisir le niveau de parallélisme pour obtenir le taux d'accélération souhaité. Faire varier le niveau de parallélisme permet de générer un ensemble d'implémentations de la même tâche mais avec différentes caractéristiques de temps d'exécution et de surface. Comme abordé dans l'état de l'art, l'exploitation du parallélisme permet de réduire la consommation énergétique et une manière d'exploiter le parallélisme est

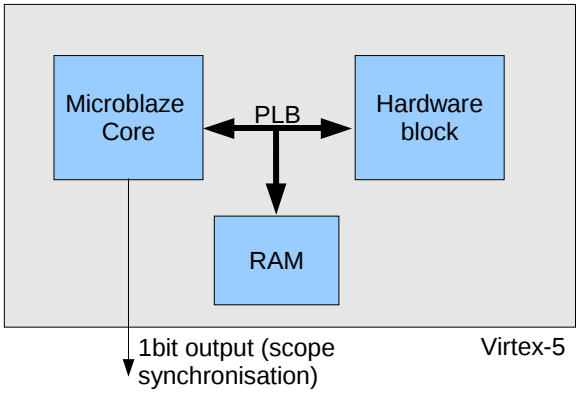

FIGURE 2.2: Contenu du FPGA pour l'expérimentation.

d'utiliser le déroulage de boucle en synthèse de haut niveau. Dans notre cas, le code C est utilisé pour la description des applications et l'outil de synthèse de haut niveau *Mentor CatapultC* est choisi pour générer les accélérateurs matériels. *CatapultC* permet de dérouler les boucles pour favoriser l'augmentation du niveau de parallélisme. Comme premier test, nous utilisons la multiplication matricielle, équation (2.1), qui est hautement parallélisable.

$$(A \times B)_{i,j} = \sum_{k=1}^{n} A_{i,k} \times B_{k,j} \tag{2.1}$$

Grâce à ses trois boucles imbriquées offrant un grand nombre d'opérations (multiplication - accumulation) et un nombre fixe d'itérations, cet algorithme est représentatif d'un traitement parallélisable et offrira une différence importante entre la solution séquentielle (plus lente) et une solution parallèle (la plus rapide) ainsi que plusieurs compromis entre ces deux extrêmes pour la construction d'un modèle de consommation énergétique en fonction des performances temporelles.

Ce travail compare et caractérise le temps d'exécution et la puissance consommée entre différents niveaux de déroulage de boucle. L'architecture est composée d'un *Xilinx MicroBlaze* (cœur logiciel) connecté à un bloc accélérateur matériel via le bus PLB, schématisé Figure 2.2. La fréquence d'horloge est de $100MHz$. La matrice est de taille 32×32 et les données sont codées sur 8 bits. Pour améliorer les transferts de données sur le bus PLB, celui-ci étant sur 32 bits, les données sont empaquetées sur 32 bits. Le code C de la multiplication matricielle est présenté Figure 2.3. Les lignes $7, 8$ et 22 sont des opérations de pré et post traitement pour dépaqueter et ré-empaqueter les données. La matrice de résultat est initialisée dans une boucle séparée, avant d'entamer les opérations, pour permettre une action du déroulage de boucles uniquement sur les opérateurs. Les deux boucles internes, lignes 17 et 18, sont inversées pour explicitement faciliter le parallélisme au niveau de la boucle in-

```
1  void matrix_mult(unsigned int A[M>>2][N],
2                   unsigned int B[N>>2][P],
3                   unsigned int C[M>>2][P]){
4      unsigned char _A[M][N], _B[N][P], _C[M][P];
5
6      //unpack inputs
7      unpack_mat(A, _A);
8      unpack_mat(B, _B);
9
10     // initialization
11     for (i = 0; i < M; i++)
12         for (j = 0; j < P; j++)
13                 _C[i][j] = 0;
14
15     // matrix multiplication
16     for (i = 0; i < M; i++)                  //loop 1
17         for (k = 0; k < N; k++)              //loop 2
18             for (j = 0; j < P; j++)          //loop 3
19                 _C[i][j] += _A[i][k]*_B[k][j];
20
21     // pack results
22     pack_mat(_C, C);
23 }
```

FIGURE 2.3: Code C de la multiplication matricielle. Le résultat de $A \times B$ est sauvegardé dans C.

térieure. La fonction `matrix_mult()` est synthétisée en utilisant *CatapultC*. Elle est ensuite encapsulée dans une interface compatible avec le bus système (PLB) pour permettre une connexion avec le *MicroBlaze*, en utilisant un gabarit fournit par Xilinx. Ce gabarit est modifié pour permettre la gestion des mémoires et registres de l'accélérateur, puis l'ensemble de l'accélérateur est généré en utilisant *Mentor Precision*. Finalement le système complet est synthétisé avec EDK et les outils de synthèse fournis par Xilinx. Cette méthodologie de génération des accélérateurs matériels est issue et décrite dans [82].

La Figure 2.4 est une mesure de consommation du cœur du FPGA pendant l'exécution de la multiplication matricielle pour deux implémentations différentes, une solution où la multiplication matricielle s'exécute sur le *MicroBlaze* et une solution où elle s'exécute sur un accélérateur matériel dédié, contrôlé par le *MicroBlaze*. Lors de l'exécution il y a premièrement la génération des deux matrices, puis ces matrices sont empaquetées dans des mots de 32 bits. La première implémentation est exécutée, il s'agit donc d'une exécution logicielle sur le processeur (*Software Execution*, SE). Ensuite la même opération s'exécute sur un accélérateur matériel (*Hardware Execution*, HE). Finalement, la matrice de résultat est dépaquetée pour être utilisée. Sur cette figure, on peut noter une différence importante du temps d'exécution (facteur d'accélération proche de 10) et de la puissance consommée entre une exécution logicielle et matérielle du même algorithme.

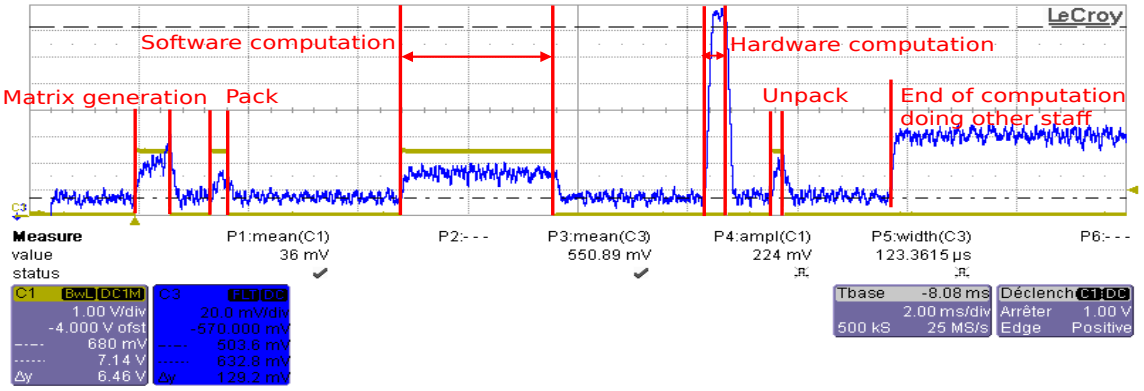

FIGURE 2.4: Exemple de mesure de puissance du cœur du FPGA pendant l'exécution de la multiplication matricielle sur le processeur puis sur un accélérateur matériel.

La section suivante présente une étude complète de la surface, la puissance consommée et le temps d'exécution des différentes implémentations matérielles de l'algorithme.

2.3 Analyse des mesures et extraction du modèle

Puisque nous nous intéressons au temps d'exécution et à la consommation en énergie en fonction de plusieurs versions d'un accélérateur matériel en faisant varier le déroulage de boucle, le paramètre principal commun aux mesures est l'indice de déroulage de boucle (LUI - *Loop Unrolling Index*). Un LUI de 1 signifie qu'il n'y a pas de déroulage de boucle, un LUI de 2 signifie que la boucle est déroulée une fois, donc que les opérateurs de la boucle sont doublés. Ceci signifie que pour un LUI de 32 avec une boucle de 32 itérations (notre cas) la boucle est complètement déroulée et toutes les opérations s'exécutent en parallèle (si la dépendance des données le permet). Chaque version du bloc matériel généré est nommée par ses paramètres de LUI pour chaque boucle. Dans l'exemple de la multiplication matricielle, trois boucles sont présentes, la notation des LUIs est présentée sous la forme $LUI_{loop1}, LUI_{loop2}, LUI_{loop3}$ et puisque les trois boucles sont imbriquées, les LUIs sont multipliés pour calculer le LUI total :

$$Total\ LUI = \prod_{i=1}^{3} LUI_{loop\ i} \tag{2.2}$$

Le LUI total peut avoir la même valeur pour différentes solutions de parallélisme en fonction du déroulage de la boucle interne ou externe. Dans cas cas, pour un même LUI total le résultat de chacune des solutions n'est pas le même. La plupart des combinaisons de LUI sont générées et testées. Les valeurs des meilleures solutions générées sont reportées dans la table 2.1 ainsi que la solution logicielle. On peut noter

TABLE 2.1: Temps d'exécution pour différents LUI, notés sous la forme LUI_{loop1}, LUI_{loop2}, LUI_{loop3}.

Versions de déroulage de boucle	logicielle	1,1,1	1,1,8	1,1,16	8,1,32	1,8,32	4,4,32
Temps d'exécution (ms)	10.75	1.04	0.51	0.47	0.42	0.38	0.39
Taux d'accélération (ref logicielle)	1	10.33	21.08	22.87	25.75	28.03	27.55
Taux d'accélération (ref 1,1,1)	0.097	1	2.04	2.21	2.48	2.74	2.67
Surface (slices)	-	112	136	153	315	555	661
Énergie (μJ)	244.79	61.99	27.34	27.55	26.61	27.48	31.58
Gain énergétique (ref logicielle)	1	3.95	8.95	8.89	9.01	8.91	7.75
Gain énergétique (ref 1,1,1)	0.25	1	2.27	2.25	2.33	2.26	1.96

que l'exécution matérielle sans déroulage de boucle est plus rapide, d'un facteur 10 dans cet exemple, que l'exécution logicielle avec un gain énergétique proche de 4. En utilisant le déroulage de boucle, les taux d'accélération peuvent atteindre 28 et un gain énergétique de 9. L'accélération matérielle apporte un gain important de performances temporelles et énergétiques par rapport à l'exécution logicielle et on note également des variations importantes entre les solutions matérielles elles-mêmes grâce à l'exploitation du déroulage de boucles. Les sections suivantes présentent les analyses détaillées des temps d'exécution, surface et énergie en fonction du LUI.

2.3.1 Analyse du temps d'exécution

La Figure 2.5 présente le temps d'exécution en fonction du LUI total et montre que le temps d'exécution diminue rapidement dès les premiers déroulages de boucle. La version avec un déroulage de boucle de huit conduit à dupliquer le cœur de boucle pour en obtenir huit implémentations (définition de *Mentor*). Cette version avec un LUI total de huit est deux fois plus rapide que la version séquentielle. Pour des LUI plus importants, le temps d'exécution est un peu plus rapide mais tend à se stabiliser, plusieurs limites dans la réduction du temps d'exécution sont possibles telles que les dépendances des opérations, des chemins critiques et des limitations d'accès aux ressources spécifiques, mémoires ou DSPs par exemple.

On note clairement que les meilleures solutions (en terme de temps) sont celles qui ont les LUI élevés dans la boucle interne. Ceci est dû aux dépendances de données qui sont plus importantes pour les autres niveaux de boucles puisque cette boucle interne correspond aux multiplications-accumulations pour le résultat d'une ligne de la matrice C. Le résultat de ces opérations d'accumulation porte sur une ligne (indice j) de la matrice de résultat et sont indépendantes entre elles. En revanche, ce n'est pas le cas pour les accumulations avec l'indice k (la variation de k est au niveau de la boucle 2) dont le résultat ne porte que sur un seul point et forcément toutes ces accumulations sont dépendantes. Le déroulage de la boucle 2 ne peut pas engendrer

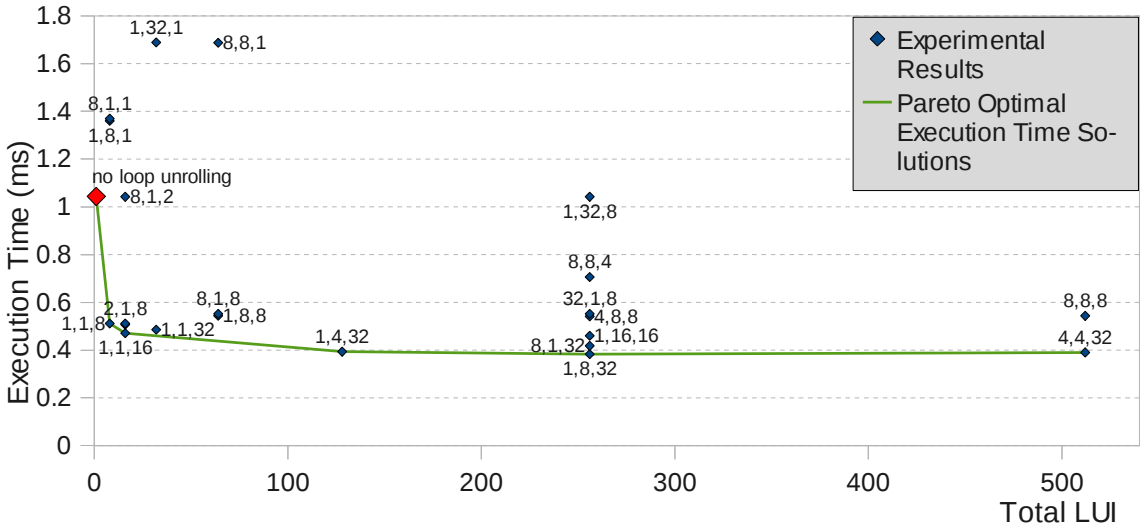

FIGURE 2.5: Temps d'exécution de plusieurs implémentations de la tâche de calcul du produit de matrice. Les différentes versions correspondent à des niveaux de déroulages de boucles différents et les LUIs de chaque boucle sont séparés par des virgules pour identifier le détail de chaque LUI, $LUI_{loop1}, LUI_{loop2}, LUI_{loop3}$. La courbe verte donne les solutions optimales pour le temps d'exécution en fonction du LUI total.

directement de parallélisme. Le déroulage de la boucle externe (boucle 1, ayant pour indice i) augmente la complexité du contrôle de l'application avec duplication de ses deux boucles internes.

Un déroulage de boucle d'un facteur huit de la boucle interne ne correspond pas à un fonctionnement huit fois plus rapide qu'une version séquentielle (sans déroulage de boucle, $Total\ LUI = 1$). Ceci est certainement dû au fait que le parallélisme des opérations est limité par le parallélisme des accès aux données, chaque matrice étant stockée dans une mémoire simple port. De plus la gestion des boucles peut occasionner des délais qui ne sont pas réduits avec le déroulage des opérations d'une boucle.

Pour comparaison avec une exécution logicielle, la même opération sur un *MicroBlaze* à 100 MHz prend 10.75 ms, alors la version séquentielle de l'accélérateur matériel procure une accélération de 10 (1.04 ms, $Total\ LUI = 1$) et peut monter à 28 (0.38 ms, $Total\ LUI = 256$) en utilisant le déroulage de boucle. Le déroulage de boucle permet un gain de temps d'exécution de 0.66 ms soit un taux d'accélération de 2.7 par rapport à une multiplication matricielle matérielle sans déroulage de boucle.

Dans la Figure 2.5, on peut voir des versions pour lesquelles le temps d'exécution est plus faible que la solution séquentielle. Ces solutions sont des solutions non optimales en terme de compromis performance-énergie-surface, par exemple une surface

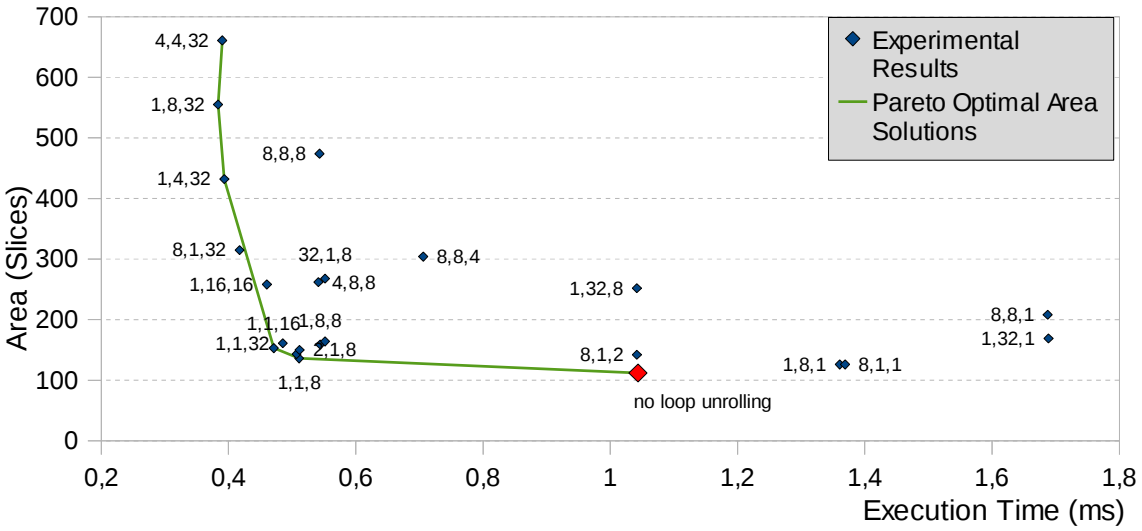

FIGURE 2.6: Surface de l'accélérateur matériel en fonction du temps d'exécution pour différents LUI. La courbe verte donne les solutions optimales pour la surface en fonction du temps d'exécution.

élevée et consommation importante pour un temps d'exécution supérieur ou identique à d'autres solutions ayant des surfaces et consommations plus faibles. Dans ce cas de figure, les déroulages de boucles complexifient l'accélérateur sans permettre d'améliorer le temps d'exécution. Ces solutions ne sont pas souhaitées lorsque l'on utilise le déroulage de boucle et ne seront pas retenues par la suite.

2.3.2 Analyse de la surface

La Figure 2.6 représente la surface pour les différentes versions synthétisées en fonction du temps d'exécution. On peut voir qu'il y a une très bonne diminution du temps d'exécution pour une faible augmentation de la surface pour le premier déroulage de boucle $(1, 1, 8)$. Pour des déroulages de boucle plus importants, la surface augmente rapidement mais le temps d'exécution reste bloqué à environ $0.4 - 0.5\ ms$, principalement à cause des accès aux données.

Ceci veut dire qu'il ne faut pas essayer d'obtenir le niveau de parallélisme maximum si la vitesse de l'algorithme est limitée par l'accès aux données. Cette limite est dépendante de la structure de l'algorithme, des mémoires de stockage et des performances des outils et méthodes d'implémentation. L'obtention de cette limite ne fait pas partie de ces travaux. De plus, les temps mesurés ici tiennent compte des temps de transferts des données vers l'accélérateur. Le transfert de données sur le bus implique un surcoût fixe et incompressible (avec la méthode appliquée ici), qui s'ajoute au temps de traitement.

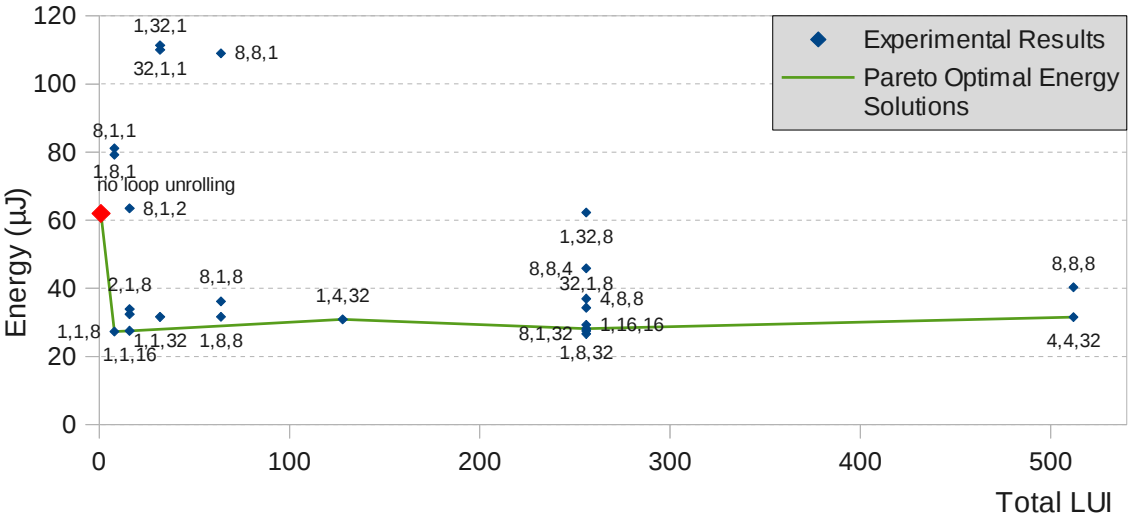

FIGURE 2.7: Énergie consommée par les différentes instances possibles de la tâche de calcul du produit de matrices en fonction du LUI total. La courbe verte donne les solutions optimales pour la consommation énergétique en fonction du LUI total.

2.3.3 Analyse de l'énergie

2.3.3.1 Énergie en fonction du déroulage de boucle

En se focalisant sur l'énergie, la Figure 2.7 montre que la consommation en énergie diminue rapidement, de 62 μJ pour la version séquentielle à 27 μJ pour une solution déroulée avec un LUI de 8. Cette valeur reste sensiblement la même avec des LUI plus importants. La courbe de l'énergie a la même forme globale que celle du temps d'exécution (Figure 2.5), on peut en déduire un lien entre la réduction d'énergie et la réduction du temps d'exécution.

À cause de la complexité et de la surface des blocs générés, nous ne sommes pas parvenu à générer un bloc avec un LUI total de plus de 512 avec *CatapultC*. Cependant, vu que les meilleures accélérations sont obtenues pour les premiers déroulages de boucles, il n'est pas nécessaire de mettre en place des déroulages de boucle importants.

Ces expérimentations montrent qu'une implémentation matérielle parallèle peut permettre d'atteindre une réduction énergétique d'un facteur 29 par rapport à une exécution séquentielle sur un processeur à fréquence de fonctionnement égale. Les consommations énergétiques observées peuvent varier, sur cet exemple, d'un facteur 2 pour les différentes solutions matérielles avec différents déroulages de boucles. Il existe donc un potentiel d'adaptation intéressant du compromis énergie-performance et surface des accélérateurs matériels. Celui-ci pourra être utilisé pour exploiter les ressources matérielles reconfigurables dynamiquement en vue d'atteindre des objec-

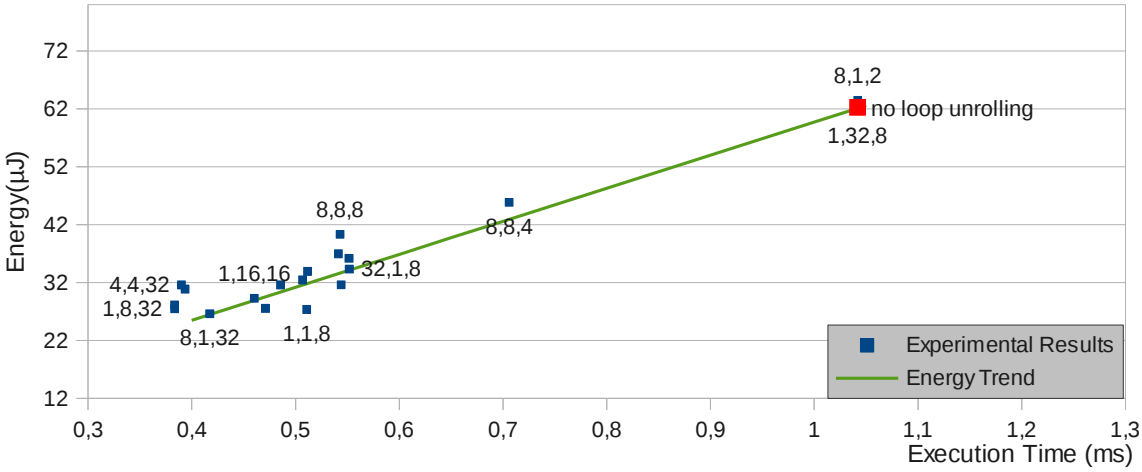

FIGURE 2.8: Énergie consommée par l'accélérateur matériel de multiplication matricielle en fonction du temps d'exécution. L'équation du modèle est l'équation (2.4).

tifs de réduction d'énergie sous contraintes de performance par exemple, ceci sera traité dans les chapitres suivants de ce mémoire. Afin de pouvoir quantifier les variations énergétiques possibles à l'aide du déroulage de boucle, nous nous intéressons à la construction d'un modèle.

2.3.3.2 Énergie en fonction du temps d'exécution

La Figure 2.8 présente l'énergie consommée par l'accélérateur matériel en fonction du temps d'exécution. Cette figure montre que l'utilisation du déroulage de boucle engendre une consommation d'énergie qui diminue linéairement avec le temps d'exécution. Cet aspect est expliqué en détail dans la section suivante.

2.3.3.3 Analyse des gains énergétiques

L'étude détaillée de la consommation au niveau de l'accélérateur matériel permet de séparer les puissances consommées par la partie contrôle de l'algorithme, la machine d'état (FSM - *Finite State Machine*), ainsi que les puissances statiques, *idle* (puissance dynamique lorsque l'accélérateur n'est pas en cours de calcul) et dynamique (lors du calcul). La Figure 2.9 est une représentation des différentes puissances consommées pour la comparaison entre deux versions d'accélérateurs matériels prises pour exemple. Les consommations de la version séquentielle et d'une version parmi les plus rapides sont représentées à l'échelle (en puissance et en temps) pour mettre en valeur les effets de variation de l'énergie.

Les valeurs de puissance et d'énergie sont rapportées dans la table 2.2. Avec ces mesures, nous pouvons vérifier que l'énergie consommée par la version parallélisée

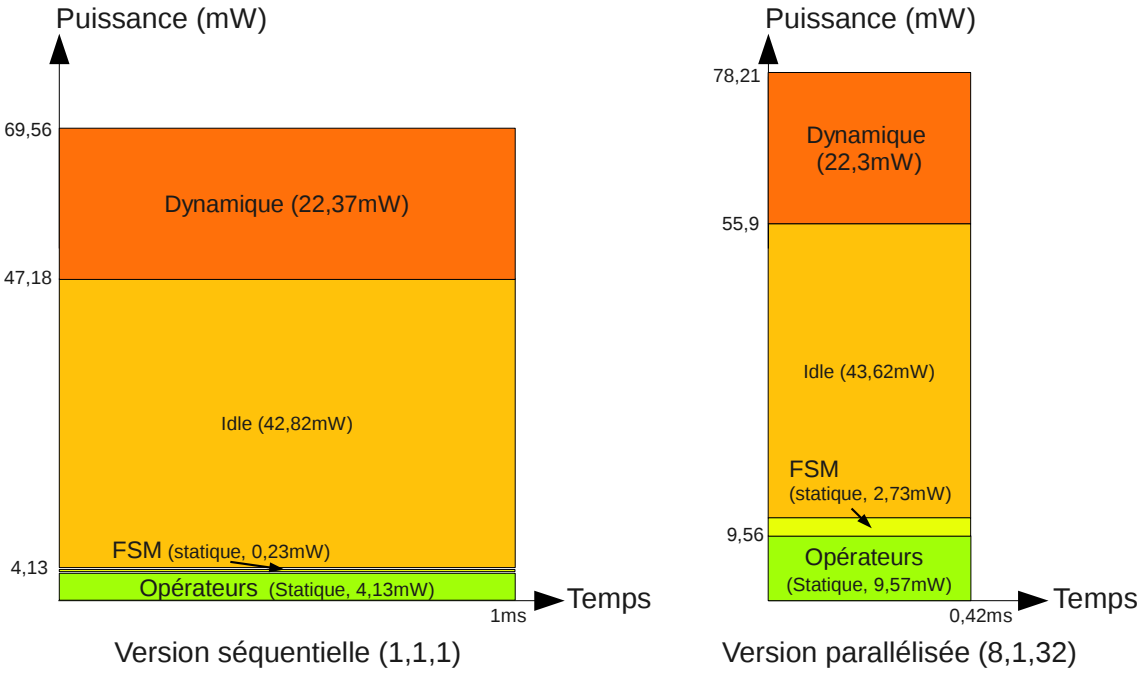

FIGURE 2.9: Représentation de la puissance consommée par deux versions accélérateurs matériels de la multiplication matricielle. Les échelles sont respectées entre les deux schémas.

est plus faible que pour la version séquentielle avec un ratio plus important pour la puissance statique, ce qui est normal vu que le parallélisme augmente la surface. On note une augmentation de la puissance statique d'un facteur 2.31 pour la partie des opérateurs et d'un facteur 11.7 pour la partie contrôle. L'augmentation importante de la surface et de la puissance consommée par le contrôle de l'accélérateur peut s'expliquer par l'augmentation du nombre d'opérateurs à contrôler ainsi qu'une gestion des données plus complexe. Cependant, grâce à l'accélération obtenue, l'énergie statique consommée par le traitement des données (opérateurs notamment) est presque constante (rapport de 0.93). En revanche on note une diminution importante de la consommation énergétique liée à la consommation *idle* et dynamique (rapport de 0.4). La réduction énergétique est du même ordre de grandeur que l'accélération engendrée. Ceci est confirmé par le fait que ces puissances respectives mesurées sont semblables entre les deux versions.

Certaines ressources sont utilisées en quantités fixes quelque soit le niveau de parallélisme. C'est le cas des BRAMs qui servent au stockage des matrices opérandes et résultats. Ces ressources provoquent une consommation de puissance *idle* constante et lorsque le temps d'exécution diminue, alors la part de l'énergie liée à ces ressources diminue proportionnellement. La consommation *idle* peut être causée en grande partie par les BRAMs. Ils sont en effet connus pour consommer environ 7 mW chacun (selon le Xilinx Power Estimator [39]) lorsque l'un des ports est activé, que celui-

TABLE 2.2: Puissances et énergies consommées par l'implémentation de deux accélérateurs matériels.

	version (1.1.1)		**version (8.1.32)**		**Ratio (%)**	
Temps (ms)	1.04		0.42		40.38	
	Puiss.(mW)	**Énergie(μJ)**	**Puiss.**	**Énergie**	**Puiss.**	**Énergie**
Statique (opérateurs)	4.13	4.30	9.56	4.01	231.13	93.34
Statique (contrôle)	0.23	0.24	2.73	1.15	1166.67	471.15
idle	42.82	44.53	43.62	16.32	101.87	41.14
Dynamique (exéc.)	22.37	23.26	22.30	9.37	99.69	40.26
Totale	69.55	72.33	78.21	30.85	112.45	44.36

ci effectue des transferts ou non. Or pour l'interface bus de l'accélérateur avec le processeur, nous avons besoin de trois mémoires tampons pour charger les données des trois matrices. Chaque mémoires est constituée d'une BRAM dans notre cas. Il est probablement possible d'optimiser l'utilisation des BRAMs afin de réduire cette proportion de puissance *idle*, mais ce n'est pas l'objectif de ces travaux.

Nous avons donc une consommation en énergie statique presque constante, une variation importante de la consommation de la partie contrôle mais dont la proportion reste faible par rapport à la consommation globale et une consommation *idle* et dynamique qui est diminuée proportionnellement au temps d'exécution. L'énergie consommée par rapport au temps d'exécution devrait suivre une droite d'équation :

$$E = \alpha + \beta t \tag{2.3}$$

avec α représentant en partie l'énergie statique des opérateurs et β représente la puissance (*idle* et dynamique) constante et dont l'énergie varie en fonction du niveau de parallélisme et du temps d'exécution.

2.3.4 Extraction du modèle

En se basant sur la Figure 2.8, nous proposons une approximation linéaire de la tendance énergétique en fonction du temps d'exécution tel que proposé dans la section précédente. L'équation affine construite est représentée sur la figure (droite verte) et est donnée par l'équation suivante

$$E\{\mu J\} = 4.3\{\mu J\} + 56\{mW\} \times t\{ms\} \tag{2.4}$$

où E est l'énergie totale en μJ et t le temps d'exécution en ms. 4.3 μJ correspond à la consommation d'énergie statique et 56 mW correspond à la puissance liée au reste des composants de l'accélérateur notamment la puissance *idle* des ressources et la puissance dynamique en cours d'exécution

Le problème de ce modèle est qu'il n'est valable que pour l'application de multiplication matricielle telle qu'elle est utilisée ici. Afin de construire un modèle plus générique qui puisse s'appliquer pour tous les accélérateurs matériels, le modèle extrait doit être adapté. La construction du modèle générique sera basée sur la mesure d'énergie et du temps d'exécution de la version séquentielle de l'accélérateur à modéliser. L'extraction du temps et de l'énergie de la solution séquentielle (1,1,1) de l'équation (2.4) permet d'obtenir une équation générique dont les paramètres peuvent être utilisés pour d'autres accélérateurs matériels :

$$E = \left(4.3 + 56 \times t \times \frac{1.04}{t_0}\right) \times \frac{E_0}{62} \tag{2.5}$$

$$E = \alpha \times E_0 + \beta \times \frac{E_0}{t_0} \times t \tag{2.6}$$

où t_0 est le temps d'exécution du nouvel accélérateur matériel avec un autre algorithme à caractériser, mesuré pour une version sans déroulage de boucle et E_0 sa consommation d'énergie. 1.04 et 62 correspondent au temps (ms) et à l'énergie (μJ) de la version séquentielle de la multiplication matricielle. L'équation peut être réduite comme proposé dans l'équation 2.6 avec $\alpha = \frac{4.3}{62}$ et $\beta = \frac{56 \times 1.04}{62} = \frac{58.24}{62}$.

La tendance de la consommation énergétique en fonction du temps d'exécution est une fonction affine où les coefficients α et β sont dépendants des performances de l'application. Ces coefficients sont évalués en mesurant uniquement les performances de la version séquentielle de l'accélérateur matériel de l'algorithme.

2.4 Applications de validation du modèle

La section précédente a montré la construction d'un modèle de consommation énergétique en fonction du temps d'exécution lors de l'utilisation du déroulage de boucle pour générer différentes versions d'un accélérateur matériel. Ce modèle est construit sur la multiplication matricielle, il est cependant admis que la multiplication matricielle peut être écrite selon plusieurs approches qui modifient ses performances, de plus cet exemple n'est pas représentatif de toutes les applications, en particulier pour des algorithmes usuels qui sont moins réguliers.

Pour valider notre approche et le modèle extrait, le déroulage de boucle est appliqué sur deux autres algorithmes : l'estimation de mouvement, utilisée dans les algorithmes de compression vidéo de type MPEG, et le filtre de déblocage, utilisé dans la norme vidéo H.264/AVC.

L'estimation de mouvement est une étape clé de la compression vidéo pour les standards basés sur MPEG. Les algorithmes de correspondance de blocs sont souvent utilisés pour calculer le vecteur de déplacement qui détermine le déplacement d'un

macrobloc de pixels donné entre deux images. L'estimation de mouvement peut être effectuée en utilisant la méthode du *Full Search* (FS). Une comparaison entre un bloc de l'image précédente et chaque position dans un macrobloc de l'image courante est calculée. Les différences se basent sur le calcul de la somme des différences absolues (SAD- *Sum of Absolute Differences*), et sont sauvegardées dans une matrice. Même si le FS n'est pas utilisé dans les applications réelles (il existe des algorithmes plus performants), son comportement est très similaire comparé à d'autres algorithmes de calcul de vecteur de mouvement.

Le filtre de déblocage (DF *Deblocking Filter*) est quant à lui appliqué sur les blocs d'une vidéo décodée pour améliorer la qualité visuelle du rendu en lissant les fronts formés par les techniques de codage par blocs. Cet algorithme fait partie des codecs H.264. Le filtre est appliqué sur les fronts verticaux et horizontaux sur les blocs *chroma* et *luma* dans un macrobloc. La force du filtre dépend de nombreux paramètres tels que le type de front, si le bloc courant est à la frontière d'un macrobloc ou non.

Comme dans le cas de la multiplication de matrices, plusieurs accélérateurs sont générés pour ces deux exemples, dans différentes configurations de parallélisme et les performances et l'énergie sont mesurées pour comparaison avec le modèle précédent.

2.4.1 Estimation de mouvement

Le temps d'exécution et la consommation d'énergie des accélérateurs du FS sont mesurés pour une taille de bloc de 4×4 pixels, une surface de recherche de 12×12 pixels et bien sûr différents LUI. Les résultats sont représentés Figure 2.10. Les mesures sur la version séquentielle donnent les résultats suivants, $E_0 = 2.01\ \mu J$ et $t_0 = 37\ \mu s$. La tendance définie dans l'équation (2.5) donne alors l'équation (2.7) qui est représentée Figure 2.10.

$$\begin{aligned} E &= \alpha \times E_0 + \beta \times \frac{E_0}{t_0} \times t \\ E_{FS} &= \frac{4.3}{62} \times 2.01 + \frac{58.24}{62} \times \frac{2.01}{0.037} \times t \\ E_{FS} &= 0.14 + 51.03 \times t \end{aligned} \tag{2.7}$$

Cette figure montre que la tendance du modèle correspond globalement aux mesures pour l'algorithme du *Full Search*. La précision du modèle est de 30% pour les LUI élevés. La réduction du temps d'exécution a le même effet sur la consommation d'énergie que pour la multiplication matricielle, une réduction de l'énergie consommée lors de l'accélération. Ces mesures montrent que pour ce second algorithme, appliquer le déroulage de boucle pour des tâches matérielles améliore l'efficacité

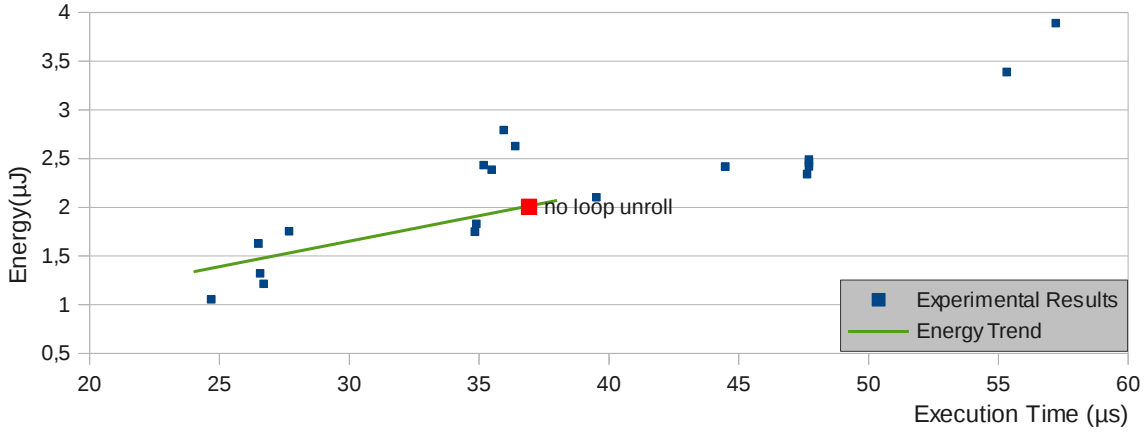

FIGURE 2.10: Énergie consommée en fonction du temps d'exécution pour l'accélérateur matériel du *Full Search*. Le modèle linéaire est donné par l'équation (2.7).

énergétique d'un facteur 1.9 et diminue le temps d'exécution d'un facteur 1.5. La consommation énergétique est de 1.05 μJ pour un temps d'exécution de 24 μs pour la solution parallèle la plus performante tandis que l'estimation du modèle donne 1.3 μJ. Cette différence entre le modèle et la mesure est probablement causée par une variation plus importante de la consommation de la partie contrôle comparée aux mesures de la multiplication matricielle. Néanmoins, malgré la très grande simplicité du modèle, la tendance énergie versus performance est bien suivie sur cet exemple.

2.4.2 Filtre de déblocage (H.264)

Pour le filtre de déblocage, la version séquentielle de l'accélérateur matériel obtient les performances mesurées de $E_0 = 3.68\ \mu J$ et $t_0 = 53\ \mu s$. En appliquant le modèle permettant d'estimer la consommation énergétique d'un accélérateur matériel avec la mise en place du déroulage de boucle, défini dans l'équation (2.5), le résultat donne l'équation 2.8. Cette estimation est représentée Figure 2.11.

$$\begin{aligned} E &= \alpha \times E_0 + \beta \times \frac{E_0}{t_0} \times t \\ E_{DF} &= \frac{4.3}{62} \times 3.68 + \frac{58.24}{62} \times \frac{3.68}{0.053} \times t \\ E_{DF} &= 0.26 + 65.22 \times t \end{aligned} \tag{2.8}$$

On peut voir que la tendance du modèle concorde avec les mesures énergétiques pour le DF. Cette courbe montre que, comme pour le précédent algorithme, une bonne utilisation du déroulage de boucle améliore l'efficacité énergétique des accélérateurs matériels. La précision du modèle est de 5% pour les versions à fort déroulage

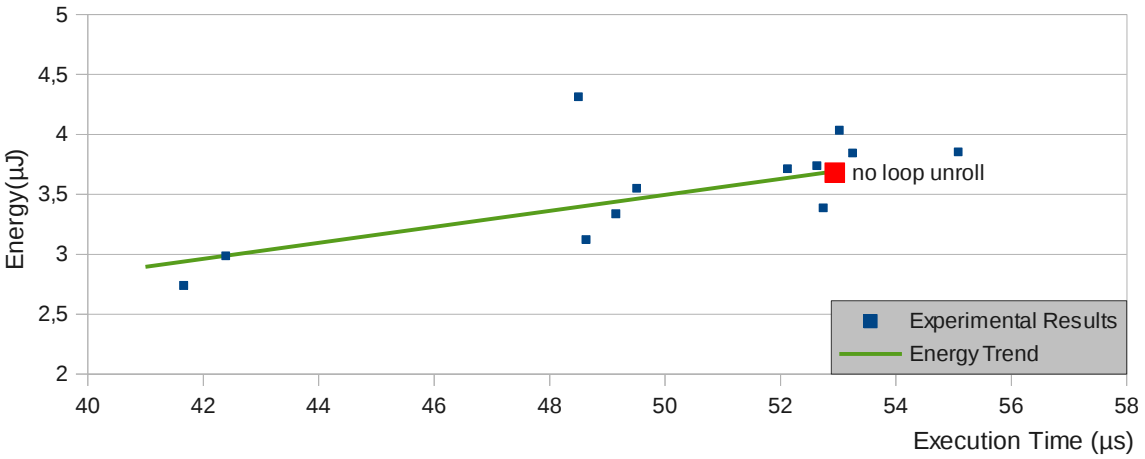

FIGURE 2.11: Énergie consommée en fonction du temps d'exécution pour l'accélérateur matériel du filtre de déblocage. Le modèle linéaire est donné par l'équation (2.8).

de boucle. Le gain énergétique maximal obtenu par déroulage de boucle est de 1.34 avec un gain en temps d'exécution de 1.26. On remarque que les accélérations obtenues ici sont beaucoup plus faibles que pour l'accélération de la multiplication matricielle. Ceci est probablement dû à la difficulté d'obtenir le parallélisme et à un contrôle des données important, il y a effectivement beaucoup d'opérations conditionnelles dans cet algorithme.

2.4.3 Analyse et résultats

Les mesures, reportées dans la table 2.3, montrent qu'il y a une très bonne accélération quand une tâche est accélérée matériellement, comparée à une exécution sur un processeur, un gain de 28.3 peut être observé pour la multiplication matricielle. Le gain en énergie est bon aussi, de 8.91 à 17.33 pour les algorithmes présentés. En exploitant le déroulage de boucle dans des outils de synthèse de haut niveau, comparé à une version matérielle séquentielle, le facteur d'accélération atteint 2.74 et un gain en énergie de 2.26. Le filtre de déblocage a les moins bons résultats des algorithmes testés. Ceci est probablement dû à une partie importante d'opérations conditionnelles et à cause de l'accès concurrent aux données. Les données sont en réalité sauvegardées dans les BRAMs du FPGA. Ce type de mémoire permet d'obtenir seulement deux données par cycle d'horloge, ce qui représente un goulot d'étranglement important lorsque les opérations sont réalisées en parallèle. Cette limitation explique partiellement pourquoi il est difficile d'obtenir un parallélisme réellement fonctionnel en parallèle et difficile d'obtenir des accélérations très importantes ainsi que les différences de performance entre les algorithmes. De plus les traitements effectués sur les données ne sont pas toujours uniquement calculatoires,

les algorithmes possèdent des structures conditionnelles qui complexifient davantage la partie contrôle de cet algorithme (FSM) et complexifient également le chemin des données ce qui peut limiter les performances.

TABLE 2.3: Comparaison du temps d'exécution et de la consommation d'énergie entre trois différentes implémentations de chaque algorithme.

	Mult. matricielle		Full Search		Filtre de Déblocage	
	Temps	Énergie	Temps	Énergie	Temps	Énergie
Soft (ms, μJ)	10.75	244.79	0.4786	18.2	0.5742	26.09
HardS (ms, μJ)	1.04	61.99	0.0369	2.01	0.0529	3.68
HardP (ms, μJ)	0.38	27.48	0.0246	1.05	0.0417	2.74
Soft/HardP **Ratio**	**28.29**	**8.91**	**19.37**	**17.33**	**13.77**	**9.52**
HardS/HardP **Ratio**	**2.74**	**2.26**	**1.50**	**1.91**	**1.26**	**1.34**

Soft représente une exécution sur le processeur *MicroBlaze*,
HardS représente une implémentation matérielle séquentielle,
HardP représente la meilleure implémentation en terme de temps.

Le modèle présenté dans ce chapitre ne permet pas de savoir s'il est possible de réduire le temps d'exécution d'un accélérateur matériel ou de combien. L'obtention de cette information nécessite une analyse très poussée de l'algorithme et la connaissance détaillée de l'outil de synthèse de haut niveau pour évaluer en particulier le niveau de parallélisme exploitable et les structures conditionnelles. Ces points sont des problématiques à part entière et ne sont pas couverts par nos travaux. Le modèle extrait ici permet de confirmer que la réduction du temps d'exécution d'un algorithme accéléré matériellement, en utilisant le déroulage de boucle, réduit la consommation énergétique proportionnellement au temps d'exécution. Selon les algorithmes, avec un parallélisme apparent ou non ou des traitements conditionnels, les valeurs d'accélération (et donc de gains énergétique) sont variables.

Les résultats montrent que la consommation énergétique diminue quand le niveau de parallélisme augmente. Pour améliorer l'efficacité énergétique, le concepteur peut explorer l'utilisation des accélérateurs matériels en profitant des possibilités d'exécution parallèle. Cependant, une tâche ayant un niveau de parallélisme plus élevé a une surface plus importante. Pour une application disposant de plusieurs tâches, si chacune de ces taches est implémentée en maximisant le parallélisme, l'implémentation totale de l'application va conduire à une surface importante et une consommation en puissance statique importante, compromettant probablement le bénéfice en énergie obtenu grâce à l'augmentation du niveau de parallélisme. Mais au cours de l'exécution d'une application, toutes les tâches ne sont pas toujours en activité simultanément. Lors des périodes de repos des tâches, il est possible, grâce à la reconfiguration dynamique, de tirer profit de cette caractéristique et de limi-

ter la surface nécessaire pour l'application en partageant les ressources matérielles. Cependant, la reconfiguration dynamique consomme de l'énergie et rend inutilisable la ressource visée pendant la reconfiguration. Il est alors important de caractériser cette fonctionnalité afin que le concepteur puisse évaluer son impact sur l'exécution complète d'une application. Ce sujet est abordé dans la section suivante.

2.5 Emploi de la reconfiguration dynamique

La reconfiguration dynamique partielle permet de changer la configuration d'une partie du FPGA, par exemple le changement de fonctionnalité d'un accélérateur matériel, pendant que le reste du FPGA continue de fonctionner normalement. Le principal avantage de cette reconfiguration est d'apporter de la flexibilité aux composants matériels puisque leur fonctionnalité peut être adaptée aux besoins. Elle permet par ce biais de diminuer la surface nécessaire puisqu'il est possible d'implémenter des accélérateurs matériels différents successivement sur la même région reconfigurable. Cependant, la reconfiguration dynamique partielle est notamment un processus de transfert de données (de configuration) et provoque une consommation d'énergie. De plus lors de la reconfiguration, la région concernée n'est pas utilisable ce qui constitue un préjudice temporel et énergétique lié à la puissance statique.

La modélisation précédente facilite l'exploration très tôt du compromis énergie et temps d'exécution des tâches matérielles. Elle montre que l'exploitation du parallélisme permet d'améliorer les performances et de diminuer la consommation énergétique. Or plus un algorithme est parallélisé, plus le nombre de ressources requises augmente et plus la reconfiguration est coûteuse. Afin d'estimer le coût complet des tâches reconfigurables dynamiquement et effectuer les choix d'implémentation, il faut également prendre en compte l'influence de la reconfiguration dynamique. Pour quantifier le coût d'utilisation de la reconfiguration dynamique et des choix d'implémentation des accélérateurs matériels, il est nécessaire de procéder à une étape de modélisation. La reconfiguration dynamique permet également d'effacer la configuration d'une région et ceci a pour but de réduire sa consommation lorsque l'accélérateur présent n'est plus utilisé. La section suivante présente une modélisation formelle de l'utilisation de cette fonctionnalité. L'objectif est d'inclure ce modèle dans la phase d'exploration afin de prendre en compte son impact dans l'exécution d'une application.

2.5.1 Modélisation du FPGA

La ressource reconfigurable du FPGA est supposée composée de N^{PRR} régions reconfigurables (PRR – Partial Reconfigurable Regions). Les PRRs sont définies lors

de la modélisation et leur description est statique. Le FPGA est modélisé comme un ensemble de PRRs et défini par

$$\mathcal{FPGA} = \{PRR_j\} \qquad \forall\, j = 1, \ldots, N^{PRR}. \tag{2.9}$$

Une application est définie par ses tâches, ordonnées selon un graphe de tâches contenant N^T tâches

$$T_i \qquad \forall\, i = 1, \ldots, N^T, \tag{2.10}$$

avec T_i la $i^{ième}$ tâche de l'application.

L'utilisation de la reconfiguration dynamique pour une application nécessite un mécanisme de gestion et de contrôle de la reconfiguration en fonction des tâches à exécuter pour assurer le bon fonctionnement du système. Le fonctionnement de ce système, qui peut être supervisé par un ordonnanceur de système d'exploitation, n'est pas détaillé ici mais nous supposons que son fonctionnement est basé sur une période de temps t_{sched} régulière. Le rôle de cet ordonnanceur est donc de fournir les tâches à exécuter en fonction du graphe de tâches de l'application et de choisir leur implémentation pour chaque intervalle de réveil t de l'ordonnanceur.

2.5.2 Modélisation des tâches et de la reconfiguration

Comme nous l'avons vu précédemment dans ce chapitre, le temps d'exécution de chaque tâche dépend de son implémentation. Ce temps est défini par la variable $C_{i,j}$ pour l'implémentation de la tâche T_i sur la PRR PRR_j. Cette valeur est placée dans $C_{i,j}(t)$ lors du début de l'exécution de la tâche correspondante. Cette dernière variable est décrémentée par l'ordonnanceur à chaque intervalle si la tâche T_i est implémentée dans l'intervalle courant, t de durée t_{sched} et représente le temps d'exécution restant de cette tâche.

La consommation de chaque tâche T_i est aussi dépendante de son implémentation, au même titre que le temps d'exécution, cette puissance est définie par la variable $P_{i,j}$ pour la tâche T_i implémentée sur la PRR PRR_j et $Pidle_{i,j}$ sa consommation *idle*, c'est-à-dire la consommation lorsque la tâche est implémentée mais n'exécute pas de calcul.

En considérant que la reconfiguration dynamique est principalement une opération de transfert des données de configuration, nous supposons que le temps nécessaire pour configurer une tâche dépend de la taille de sa configuration et de la performance du transfert de données. La puissance lors de la reconfiguration est quant à elle liée à la performance du transfert des données et du type de contrôleur de reconfiguration. Le temps et la puissance nécessaires pour configurer la PRR PRR_j

sont respectivement donnés par les variables T_{PRRj} et P_{PRRj}.

2.5.3 Modélisation de la consommation énergétique

À partir de ces définitions, nous proposons de modéliser la minimisation de l'expression de l'énergie du système global. Le but de l'optimisation est d'obtenir les configurations de tâches qui permettent de limiter la consommation d'énergie du système pour chaque intervalle t. L'instanciation de ces tâches est définie par la variable $X_{i,j}(t)$ dont la valeur vaut 1 si la tâche T_i est instanciée sur la PRR PRR_j à l'intervalle t, sinon elle vaut 0. Pour chaque intervalle t, l'ordonnanceur fournit les tâches à configurer et sur quelles PRRs elles doivent l'être. Étant donné que l'ordonnanceur doit produire les valeurs $X_{i,j}(t)$ pour chaque intervalle t de l'ordonnancement, les implémentations lors du précédent intervalle $X_{i,j}(t-1)$ sont connues. Les informations de l'intervalle précédent servent à connaître quels accélérateurs sont déjà configurés et à déterminer lesquels doivent être configurés pour satisfaire l'exécution pour l'intervalle courant t.

La puissance consommée lors de cet intervalle t peut être découpée en plusieurs contributions. Premièrement, il faut considérer la puissance consommée par les tâches qui sont déjà implémentées $P_{prev}(t,j)$, équation (2.11). Cette équation est constituée de la puissance des tâches implémentées à l'intervalle t et déjà présentes pendant l'intervalle $t-1$ donc $X_{i,j}(t) \times X_{i,j}(t-1) = 1$. Deuxièmement, une tâche configurée sur une PRR consomme de l'énergie même lorsqu'elle n'est pas en cours d'exécution. Pour réduire cette consommation, il est possible d'effacer la configuration de la PRR (*blank*). Caractérisée par $P_{blank}(t,j)$ et donnée par l'équation (2.12), elle est constituée des PRRs qui n'ont pas de tâche configurée, donc $X_{i,j}(t) = 0$ pour la PRR_j. Troisièmement, la consommation des nouvelles tâches à configurer $P_{curr}(t,j)$. Cette puissance est donc constituée de la reconfiguration pour les tâches non configurées lors de l'intervalle $t-1$ et qui doivent être configurées pour permettre l'exécution pour l'intervalle courant. Ceci tient donc compte des tâches pour lesquelles $X_{i,j}(t) = 1$ et $X_{i,j}(t-1) = 0$ et est donnée par l'équation (2.13). Finalement, il faut considérer la puissance consommée pendant l'exécution des nouvelles tâches qui viennent d'être configurées, donc pour lesquelles $X_{i,j}(t) = 1$ et $X_{i,j}(t-1) = 0$ également. Le calcul est donné par $P_{next}(t,j)$ et l'équation (2.14). Attention cependant, ces équations définissent la puissance consommée et non l'énergie et ne tiennent pas compte des temps de reconfiguration nécessaires avant l'exécution des nouvelles tâches.

$$P_{prev}(t,j) = \sum_{i=1}^{N^T} ((X_{i,j}(t) \times (X_{i,j}(t-1)) \times P_{i,j} \tag{2.11}$$

$$P_{blank}(t,j) = \sum_{i=1}^{N^T} (1 - X_{i,j}(t)) \times P_{blank,j} \tag{2.12}$$

$$P_{curr}(t,j) = \sum_{i=1}^{N^T} (X_{i,j}(t) \times (1 - (X_{i,j}(t-1))) \times P_{PRRj} \tag{2.13}$$

$$P_{next}(t,j) = \sum_{i=1}^{N^T} (X_{i,j}(t) \times (1 - (X_{i,j}(t-1))) \times P_{i,j} \tag{2.14}$$

$$\forall\ j = 1, \ldots, N^{PRR}$$

Considérant ces équations, l'énergie pendant un intervalle d'ordonnancement peut être calculée par l'équation (2.15) qui tient compte des temps de reconfiguration. Les temps de reconfiguration sont considérés comme étant plus courts que l'intervalle d'ordonnancement. Dans le cas contraire, il faut découper le temps de reconfiguration sur plusieurs intervalles d'ordonnancement.

$$\begin{aligned} E_{t_{sched}}(t) = \sum_{j=1}^{N^{PRR}} & P_{prev}(t,j) \times t_{sched} \\ & + P_{blank}(t,j) \times t_{sched} \\ & + P_{curr}(t,j) \times T_{PRRj} \\ & + P_{next}(t,j) \times (t_{sched} - T_{PRRj}). \end{aligned} \tag{2.15}$$

À partir de cette équation, le problème d'optimisation de la consommation d'énergie consiste donc à calculer les valeurs de la variable $X_{i,j}(t)$, c'est-à-dire de déterminer les choix d'implémentation des tâches sur les PRRs. Ces choix sont effectués pour l'intervalle t en fonction des tâches prêtes à être implémentées, de leur temps d'exécution $C_{i,j}(t)$ et des implémentations lors de l'intervalle précédent $X_{i,j}(t-1)$. La résolution de cette équation nécessite la connaissance de la puissance consommée lors de la reconfiguration. De plus, lorsqu'une tâche est configurée, une puissance dite *idle* est présente, même lorsque la tâche n'est pas en cours de traitement. Cette puissance peut être réduite en effectuant un effacement de la configuration.

2.5.4 Effacement de la configuration

Chaque tâche a une consommation dite *idle* qui est présente dès lors qu'elle est configurée, une PRR ayant une tâche configurée consomme plus qu'une PRR effacée.

Cette consommation est définie par la variable $Pidle_{i,j}$ représentant la consommation *idle* de la tâche T_i sur la PRR_j. Comme indiqué dans les équations précédentes, il est possible d'effacer la configuration d'une PRR en utilisant une tâche *blank*. Celle-ci permet donc de réduire la consommation lorsque la tâche précédemment configurée n'est plus utilisée. Cependant la reconfiguration de la tâche *blank* nécessite un temps de transfert des données de configuration et consomme de l'énergie. Le gain obtenu par l'effacement de la PRR peut être inférieur au coût dépensé pour la reconfiguration d'effacement.

La réduction de la consommation énergétique en utilisant l'effacement est un compromis entre le gain d'énergie obtenu en effectuant l'effacement et le surcoût engendré par cet effacement. Pour calculer le gain énergétique, il est nécessaire de connaître le temps durant lequel la PRR ne sera pas utilisée. Hormis le cas où l'ordre d'exécution est fixé à l'avance ainsi que les implémentations matérielles, l'ordonnanceur doit effectuer une prédiction (non étudiée ici). Ce temps de non utilisation de la PRR est défini par la variable $C_{blank,j}$ et le choix d'effacer ou non la PRR est défini par la variable $X_{blank,j}$ ($X_{blank,j} = 1$ pour l'effacement ou $X_{blank,j} = 0$ si l'effacement n'est pas choisi). L'équation (2.16) à minimiser est composée de trois parties. En premier l'énergie d'exécution de la tâche courante sur la PRR est donnée par l'expression (2.17). Puis l'énergie lorsqu'il y a un effacement, donc constituée d'une reconfiguration plus l'énergie consommée par la PRR une fois effacée, est donnée par l'expression (2.18). Finalement, l'énergie consommée s'il n'y a pas d'effacement, constituée de la puissance *idle* de la tâche configurée fois le temps de non utilisation de celle-ci, est donnée par l'expression (2.19). Le temps de non utilisation de la ressource est composé du temps disponible à l'effacement ($C_{blank,j}$) auquel est ajouté le temps de reconfiguration (T_{PRRj}) car il n'y a pas eu configuration de la tâche *blank*.

$$
\begin{aligned}
E_{PRR}(i,j) \quad &= && (2.16)\\
&P_{PRRj} \times T_{PRRj} + P_{i,j} \times C_{i,j} + && (2.17)\\
&X_{blank,j} \times (P_{PRRj} \times T_{PRRj} + P_{blank,j} \times C_{blank,j}) + && (2.18)\\
&(1 - X_{blank,j}) \times (Pidle_{i,j} \times (C_{blank,j} + T_{PRRj})) && (2.19)
\end{aligned}
$$

$$
\begin{aligned}
&\forall\, j = 1, \ldots, N^{PRR}\\
&\forall\, i = 1, \ldots, N^{T}
\end{aligned}
$$

Il faut noter que $C_{i,j}$ n'est pas dépendant de t ici car le temps total d'exécution de la tâche est considéré, indépendamment de l'intervalle courant de l'ordonnanceur. Il en est de même pour $X_{blank,j}$. Cette équation sert à déterminer si l'effacement de

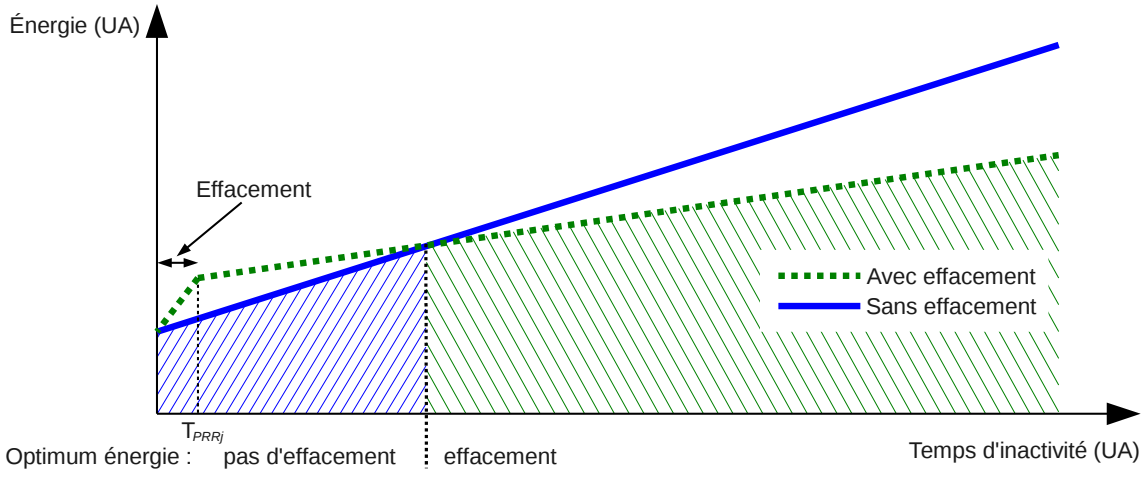

FIGURE 2.12: Représentation du compromis énergétique d'utilisation de l'effacement (représentation de l'équation (2.16) en unités arbitraires).

la PRR après l'implémentation d'une tâche est valable énergétiquement parlant. La minimisation de l'équation (2.16) s'effectue une fois la prédiction du temps de repos disponible de la PRR après effacement ($C_{blank,j}$). L'ordonnanceur doit déterminer la valeur de la variable $X_{blank,j} \in \{0,1\}$ qui réduit la consommation d'énergie lors de la configuration de la tâche i sur PRR_j. Une représentation du problème est présentée Figure 2.12 où l'on peut voir les différences d'énergie consommée par le choix d'effacement ou de non effacement en fonction du temps de repos de la ressource. L'énergie au départ est identique et correspond à l'énergie d'exécution de la tâche. La courbe en bleu représente l'évolution de l'énergie consommée dans le cas où la PRR n'est plus utilisée et n'est pas effacée. L'énergie évolue de manière constante en fonction de la consommation statique et *idle* et du temps. La courbe verte représente quant à elle l'évolution de l'énergie consommée dans le cas où la PRR est effacée. La première partie de la courbe monte rapidement à cause de la consommation de la reconfiguration d'une tâche *blank*, puis la seconde partie de la courbe (après le temps de reconfiguration T_{PRRj}) augmente moins rapidement grâce à la réduction de la consommation *idle*. Les deux courbes se croisent et ce point correspond au temps minimum d'inactivité de la PRR pour que l'effacement soit rentable du point de vue de l'énergie.

La résolution de cette équation nécessite la connaissance de la puissance et du temps de reconfiguration (et de l'énergie résultante de ces deux paramètres). Afin de pouvoir utiliser ces équations, une analyse approfondie de la reconfiguration et un modèle des coûts (temps, puissance, énergie) est nécessaire.

2.6 Conclusion

La première partie de ce chapitre met en évidence que l'exploitation du niveau de parallélisme permet d'obtenir plusieurs versions d'accélérateurs matériels pour un même algorithme et que ces versions ont diverses caractéristiques de temps d'exécution, d'énergie et de surface. Un modèle simple a été proposé mettant en évidence que la consommation énergétique diminue linéairement lorsque le temps d'exécution diminue, grâce à une bonne exploitation du parallélisme. Ces résultats sont utilisés dans l'exploration à haut niveau en particulier pour les choix d'implémentation des tâches sur les ressources d'exécution et l'estimation de l'énergie consommée. De plus, l'utilisation de la synthèse de haut niveau dans le flot de conception permet d'obtenir les différentes versions d'un accélérateur rapidement.

La seconde partie concerne l'utilisation de la reconfiguration dynamique partielle pour l'implémentation des accélérateurs matériels. Une modélisation de l'exécution des tâches est exprimée et celle-ci montre qu'un modèle de consommation de la reconfiguration dynamique doit être inclus pour tenir compte de l'ensemble des paramètres. L'utilisation de l'effacement pour réduire la consommation nécessite un modèle précis de la consommation de la reconfiguration afin de quantifier le bénéfice ou la perte du point de vue énergétique. De plus, l'effacement permet de réduire la consommation *idle* de la PRR considérée. Au cours de la reconfiguration, la proportion de la consommation *idle* du FPGA varie nécessairement. À notre connaissance, aucun modèle de consommation de la reconfiguration dynamique partielle ne mentionne ce phénomène dans l'état de l'art. Pour cette raison, la suite de ces travaux se poursuit sur l'analyse de la consommation lors de la reconfiguration et la construction d'un modèle. L'objectif de cette modélisation est d'offrir à l'ordonnanceur les moyens d'évaluer l'impact des choix d'implémentation, d'allocation et d'ordonnancement qu'il devra effectuer au cours du temps.

3 Analyse de la Consommation de la Reconfiguration Dynamique

Contenu

3.1 Introduction

La reconfiguration dynamique permet d'apporter de la flexibilité aux architectures matérielles et permet un meilleur taux d'utilisation de la surface disponible, cela s'accompagne donc d'une possible réduction de la surface. Cette réduction de la surface requise pour exécuter la même application permet de réduire les coûts de production et de réduire la consommation statique en vue d'obtenir une réduction globale de la consommation énergétique. Cependant la reconfiguration dynamique introduit de nouvelles possibilités d'ordonnancement qui doivent être gérées en ligne, la reconfiguration a alors un impact énergétique qui peut influencer les choix d'ordonnancement effectués (en ligne ou non). Pour effectuer des choix qui réduisent la consommation, l'ordonnanceur doit s'appuyer sur un modèle de la consommation de la reconfiguration dynamique. La précision du modèle influencera la pertinence des implémentations et ordonnancements choisis en vue d'atteindre les objectifs de consommation ou de temps d'exécution notamment.

Les choix d'implémentation disponibles pour l'ordonnanceur sont :

- l'implémentation d'une tache matérielle statique, sans reconfiguration dynamique,
- l'implémentation d'une tache matérielle dynamique, reconfigurée dynamiquement selon le besoin,
- l'implémentation d'une tache logicielle, fonctionnant sur un processeur.

Afin d'effectuer le bon choix, chaque solution doit pouvoir être évaluée. Notons de plus que, comme nous l'avons présenté dans le chapitre 2, une tâche peut avoir plusieurs instances matérielles différentes offrant des performances différentes.

L'évaluation des choix passe par une analyse fine de leur impact. Pour ce qui concerne la mise en œuvre de la reconfiguration dynamique, il est absolument nécessaire de prendre en compte le temps de cette opération et l'énergie consommée, d'autant plus que pendant ce temps cette région n'est pas disponible. La reconfiguration a un impact sur les performances générales de l'application, impact que nous cherchons à quantifier.

Si la littérature aborde très largement la problématique de la reconfiguration dynamique, il existe très peu d'études couvrant la caractérisation de la consommation de cette opération. Aussi, il est aujourd'hui difficile d'appréhender l'impact et l'apport de la reconfiguration dynamique (par rapport à une solution statique, sans reconfiguration dynamique) sur la puissance consommée par un FPGA.

Ce chapitre présente en premier lieu une analyse théorique de la consommation lors de la reconfiguration, trois modèles sont proposés et discutés. Le chapitre se poursuit par une analyse et une modélisation précise de la consommation de la reconfiguration dynamique. Nous montrons comment ce modèle permet d'obtenir une estimation précise du gain apporté par la mise en place de ce concept lors de la conception d'un système. Finalement, nous discutons de l'intérêt des trois modèles proposés, et justifions le choix de l'un d'entre eux pour la suite de nos travaux.

3.2 Modèles de consommation

La reconfiguration dynamique consiste à transférer des données depuis une mémoire de stockage vers la mémoire/matrice de configuration du FPGA et la mise en œuvre de cette configuration. La puissance liée à cette opération est alors principalement dynamique.

La puissance *idle* du FPGA est la puissance consommée par le FPGA lorsque celui-ci est alimenté, configuré et en fonctionnement, mais qu'aucun calcul n'est effectué. Cette puissance est composée de la puissance statique du FPGA ainsi que de la puissance dynamique liée au routage de l'horloge. Lors de l'exécution d'un calcul, la puissance dynamique liée à ce calcul (ne faisant pas partie de la consommation *idle*) est ajoutée pour former la puissance totale du FPGA. La Figure 3.1 illustre cette définition.

Une manière intuitive de représenter consommation de la reconfiguration dynamique est de prendre la consommation générale du FPGA en *idle*, P_{FPGA}^{idle}, et de lui ajouter la consommation dynamique de la reconfiguration. Cette puissance consom-

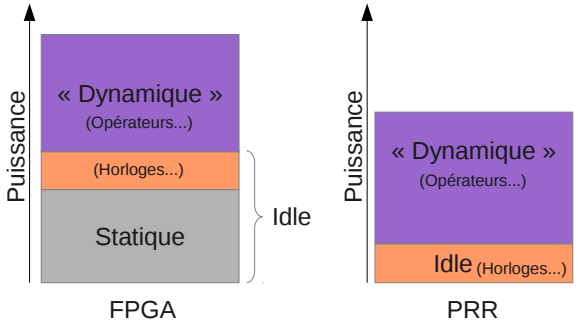

FIGURE 3.1: Représentation de la puissance *idle*.

mée durant la reconfiguration est mesurée sur plusieurs reconfigurations, soustraite à la consommation du FPGA en *idle* et moyennée. Cette valeur est considérée comme la puissance consommée par le contrôleur de reconfiguration et est définie par $P_{controller}$ dans la suite.

L'équation définissant ce modèle est

$$P = P^{idle}_{FPGA} + P_{controller} \tag{3.1}$$

et une représentation de la puissance obtenue par ce modèle est présentée Figure 3.2. Cette figure est une représentation de la consommation en puissance de l'exécution séquentielle de deux tâches matérielles sur une même PRR, séparées par une reconfiguration dynamique, modélisée par l'équation 3.1.

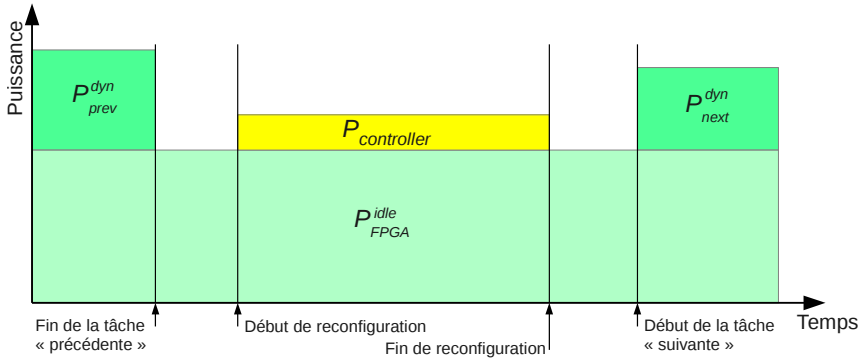

FIGURE 3.2: Représentation intuitive du profil de consommation de la reconfiguration dynamique, selon l'équation 3.1.

La consommation du FPGA en *idle* P^{idle}_{FPGA} est présente tout le temps de l'exécution et elle est constante, la consommation dynamique de la première tâche, notée P^{dyn}_{prev}, s'y ajoute au début, suivie par un instant d'inactivité. Enfin, la reconfiguration a lieu représentée par $P_{controller}$, suivie d'un instant d'inactivité, finalement la seconde tâche s'exécute avec une puissance notée P^{dyn}_{next}. Les instants d'inactivité sont représentés pour mieux visualiser les consommations en jeu et mettre en évidence la

puissance *idle*. Pendant ces instants, ni le processeur ni les accélérateurs matériels n'effectuent de traitement.

Notons qu'un modèle de ce type permet d'obtenir facilement une estimation énergétique de la reconfiguration dynamique, il suffit de considérer la puissance moyenne du contrôleur de configuration et sa vitesse de reconfiguration. Le résultat, en Joules par octet de reconfiguration (J/B), peut être utilisé dès que la taille du *bitstream* est connue.

Or la consommation *idle* de la PRR dépend de la configuration des bascules et notamment du routage des horloges. La représentation de la consommation de la reconfiguration dynamique est ici imprécise par le fait que la puissance *idle* de la PRR dépend de la configuration actuelle. En conséquence, si on considère le modèle défini par l'équation 3.1, ceci signifie qu'en pratique la puissance *idle* du FPGA varie.

3.2.1 Modèle à gros grain

La puissance *idle* de la PRR varie en fonction de sa configuration. Pour éviter les variations de la consommation *idle* globale du FPGA, nous distinguons une composante de la puissance *idle* due à la configuration de la PRR seule, indépendamment du reste du FPGA. Ainsi un premier modèle, appelé modèle à gros grain, considère la puissance *idle* du FPGA ainsi que la puissance *idle* de la PRR précédemment configurée, notée P^{idle}_{prev}. La puissance *idle* du FPGA P^{idle}_{FPGA}, ne correspondant plus à la consommation totale du FPGA mais désormais, à la puissance *idle* sans considérer la PRR (ou considérant la PRR effacée). Puis à la fin de la configuration, naturellement, la puissance *idle* de la PRR passe à sa nouvelle valeur correspondant à la nouvelle configuration, P^{idle}_{next} (non représentée dans le modèle car elle intervient à la fin de la configuration).

Ce modèle est défini par l'équation

$$P = P^{idle}_{FPGA} + P^{idle}_{prev} + P_{controller} \tag{3.2}$$

et la représentation de ce modèle Figure 3.3, montre bien le même type de modèle que l'équation 3.1 sauf que la consommation *idle* de la PRR précédemment configurée est prise en compte.

3.2.2 Modèle à grain moyen

Le comportement en consommation (*idle*) de la PRR et du FPGA entier lors de la reconfiguration doit passer de la puissance de la tâche configurée précédemment, notée P^{idle}_{prev}, à la puissance de la tâche configurée postérieurement, notée P^{idle}_{next}. Dans le modèle précédent, le changement se fait brutalement à la fin de la reconfiguration.

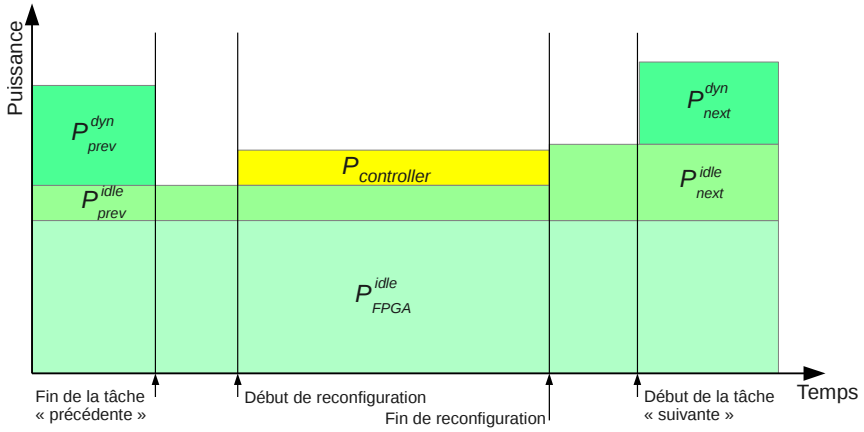

FIGURE 3.3: Représentation intuitive du profil de consommation de la reconfiguration dynamique en considérant la consommation de la PRR, selon l'équation 3.2.

En supposant que la configuration est mise en place de manière continue et progressive durant la phase de reconfiguration, le passage de P^{idle}_{prev} à P^{idle}_{next} peut être estimé par une interpolation linéaire.

Ce modèle, appelé modèle à grain moyen, est défini par :

$$\begin{aligned} P(t) &= P^{idle}_{FPGA} + P^{idle}_{prev} + P_{controller} \\ &+ (P^{idle}_{next} - P^{idle}_{prev}) \times \frac{t}{BS_{size} \times T_{1word}} \end{aligned} \tag{3.3}$$

où t est le temps, BS_{size} est la taille du *bitstream* en octets et T_{1word} est le temps requis pour configurer un mot de configuration du *bitstream* (32 *bits*). Ces deux derniers paramètres sont utilisés pour calculer l'équation de l'interpolation en fonction du débit de reconfiguration et de la taille du *bitstream*. La représentation de ce modèle est tracée Figure 3.4. Le scénario est le même que pour les figures précédentes, l'exécution séquentielle de deux tâches sur la même PRR, séparées par une reconfiguration dynamique. Cette fois la consommation *idle*, qui peut être vraiment différente pour chaque tâche est prise en compte et interpolée linéairement sur la durée de la reconfiguration dynamique.

Dans l'hypothèse où la consommation *idle* est modifiée linéairement, l'interpolation présentée permet une approche proche du comportement attendu lors de la reconfiguration dynamique. L'interpolation des puissances *idle* sera alors considérée comme puissance de référence pour le modèle de variation de la consommation d'une reconfiguration dynamique. Comme la puissance *idle* varie selon les tâches configurées, cette référence est nécessaire afin de déterminer une puissance et une énergie de la reconfiguration qui soit réaliste.

Les modèles présentés précédemment permettent une approche à deux niveaux de

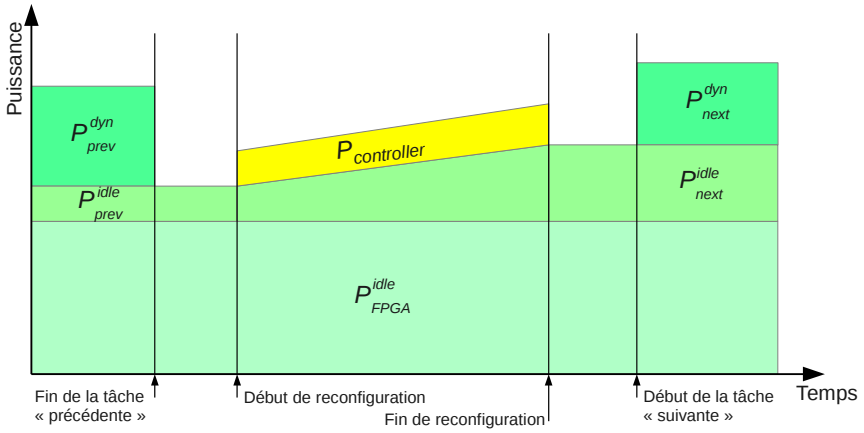

FIGURE 3.4: Représentation du profil de consommation de la reconfiguration dynamique en tenant compte de la variation de la consommation *idle* de la PRR, selon l'équation 3.3.

finesse pour l'estimation de la consommation de la reconfiguration dynamique. Le premier, modèle à gros grain défini par l'équation 3.2, est construit avec uniquement une mesure moyenne de la puissance consommée, le second, modèle à grain moyen défini par l'équation 3.3, prend en compte la variation de la consommation *idle* au cours de la reconfiguration. La première équation présentée, équation 3.1, n'est pas considérée comme un modèle utilisable. La puissance *idle* du FPGA dépend de sa configuration et l'utilisation de l'équation avec le paramètre P^{idle}_{FPGA} est ambiguë. Le modèle à gros grain sépare la puissance *idle* de la PRR du FPGA et son utilisation est préférable.

L'estimation précise de la consommation lors de la reconfiguration est importante pour effectuer les choix élémentaires d'implémentation dans les systèmes où les budgets de puissance ou d'énergie sont contraints, notamment lorsque les temps d'exécution des tâches d'une application ou leur puissance sont proches de ceux de la reconfiguration dynamique. Un modèle précis de la consommation de la reconfiguration permettrait d'effectuer au mieux les choix d'implémentation et d'exécution des tâches matérielles pour réduire l'énergie ou la puissance requise. La section suivante présente les procédures mises en place pour confronter ces modèles théoriques de consommation aux mesures réelles afin de quantifier leur précision.

3.2.3 Étude de la précision

La reconfiguration dynamique est principalement disponible sur les FPGAs produits par Xilinx. En conséquence les travaux de reconfiguration dynamique sont effectués sur la série Virtex des FPGAs Xilinx, en particulier le Virtex-5.

Le Virtex-5 est organisé en domaines d'horloge. Le XC5VLX50T, utilisé dans ces

mesures, a six domaines d'horloges. Dans ces domaines d'horloges, la configuration est organisée en *frames*, elles-mêmes organisées en colonnes qui contiennent soit des CLBs (les slices), soit des DSPs, soit des BRAMs ou d'autres blocs spécifiques. L'architecture est présentée Figure 3.5.

La configuration d'une *frame* nécessite 41 mots de 4 octets (soit 164 octets) et le nombre de *frames* dans une colonne dépend du type de ressources présentes dans celle-ci. La région minimale adressable est une *frame* et la taille minimale recommandée pour la reconfiguration est une colonne. La région reconfigurable (PRR) doit donc être un multiple de *frames* et contient principalement des CLBs, DSPs et des BRAMs. Il est plus rare d'utiliser des blocs d'entrées-sorties ou de génération d'horloge dans une région reconfigurable sauf pour certains cas particuliers.

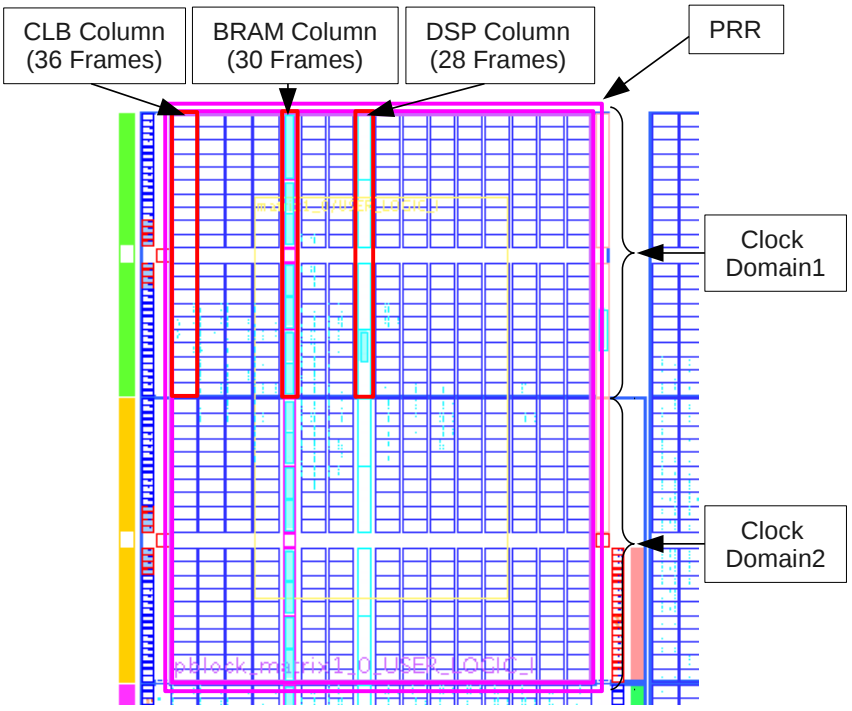

FIGURE 3.5: Capture de l'outil PlanAhead représentant environ un quart de l'architecture du Virtex-5 XC5VLX50T correspondant à la PRR sélectionnée.

3.2.3.1 Procédure de mesure

La plateforme utilisée est la même que dans le chapitre précédent avec la même procédure de mesure. Le FPGA est configuré à partir des informations du *design* de référence pour la reconfiguration dynamique fourni par Xilinx ([83]), c'est à dire que le projet est composé d'un processeur configuré (*MicroBlaze*), d'un contrôleur de mémoire CompactFlash et d'un contrôleur de reconfiguration de Xilinx (*xps_hwicap*) comme présenté Figure 3.6. Le processeur *MicroBlaze* et le contrôleur de reconfigu-

ration fonctionnent tous deux à fréquence constante de 100 MHz. Les demandes de reconfiguration sont gérées par le *MicroBlaze* qui lit alors le fichier de configuration depuis la CompactFlash puis envoie les données vers *xps_hwicap* qui va ensuite appliquer la nouvelle configuration à la PRR.

Dans les expériences suivantes, la PRR sélectionnée correspond à celle présentée au début de la section 3.2.3 et permet de contenir la plus grande tâche implémentée. La PRR correspond à un fichier de configuration de 227 700 *octets*.

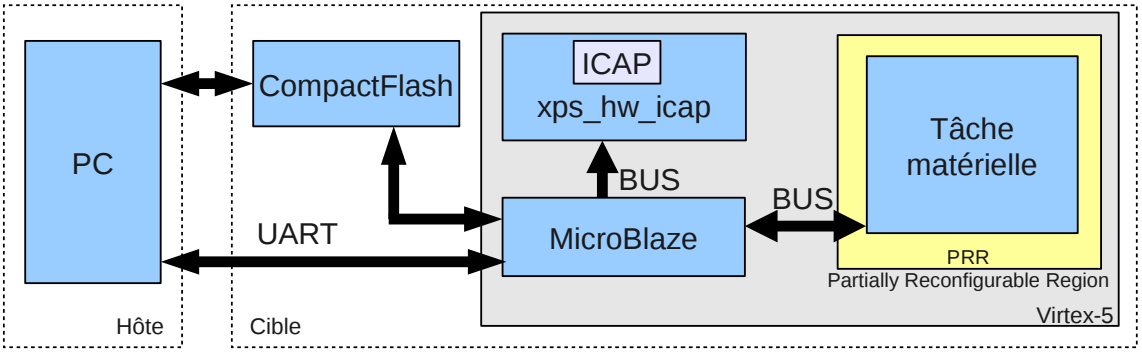

FIGURE 3.6: Contenu du FPGA pendant la procédure de mesure.

Les mesures de consommation sont faites pendant la reconfiguration d'une PRR avec trois tâches différentes. La première tâche, T_1, est une multiplication matricielle. La deuxième tâche, T_2, est aussi une multiplication matricielle mais sa version est différente en vue d'exploiter le parallélisme des opérations pour diminuer le temps d'exécution et l'énergie consommée. Ces tâches sont générées à partir d'un outil de synthèse de haut niveau en utilisant la méthode présentée dans le chapitre 2. La troisième tâche est appelée *blank* et notée T_{blank} par la suite. Un *bitstream blank* correspond à une configuration vide de la PRR, proposé par Xilinx, et peut être utilisé pour effacer la configuration et permettre de réduire la puissance consommée, ouvrant des possibilités d'optimisation de la consommation à l'aide de la reconfiguration dynamique. La puissance *idle* correspond à la puissance consommée par une tâche implémentée mais pas en cours d'exécution, c'est-à-dire inactive. Les caractéristiques des tâches, comprenant la puissance *idle* et les ressources utilisées, sont regroupées dans la table 3.1. La puissance *idle* de T_{blank} est considérée comme nulle selon la définition utilisée dans les modèles à gros grain et grain moyen. La puissance statique notamment est prise en compte avec la consommation *idle* du FPGA ceci dans le but de considérer uniquement les variations liées à la configuration de la PRR.

Afin de déterminer la précision des deux modèles de consommation de la reconfiguration dynamique définis précédemment, nous effectuons les mesures de consommation lors de la reconfiguration dans quatre cas. La reconfiguration entre les deux

TABLE 3.1: Ressources requises et puissance *idle* consommée pour chacune des configurations disponibles sur la PRR ainsi que la consommation globale du FPGA.

Tâche	BRAMs	Slices	DSPs	Puissance idle (mW)
T_1	8	154	1	24
T_2	8	169	2	26
T_{blank}	0	0	0	0
FPGA sans PRR configurée				402

tâches T_2 vers T_1, puis l'inverse, de T_1 vers T_2 puis en faisant intervenir la tâche vide, T_1 vers T_{blank} et finalement T_{blank} vers T_2. L'ensemble de ces mesures permet une bonne couverture des différentes possibilités de reconfiguration disponibles.

Les mesures dans quatre cas de reconfiguration sont effectuées et les courbes résultats sont présentées figures 3.7, 3.8, 3.9 et 3.10. Les estimations données par les deux modèles sont aussi représentées sur ces figures. Il est à noter que la puissance estimée par les modèles n'est valide que pour la durée de la reconfiguration, en dehors de celle-ci, la puissance représentée est la puissance *idle*. Ceci explique les différences entre les estimations et mesures avant et après la reconfiguration dynamique où la puissance dynamique d'exécution des tâches n'est pas modélisée.

La mesure entre la puissance *idle* avant la reconfiguration et la puissance moyenne pendant la reconfiguration donne une puissance $P_{controller}$ de 20 mW. La puissance *idle* du FPGA avec une PRR vide est donnée dans la table 3.1 et vaut $P^{idle}_{FPGA} = 402\ mW$. La taille du *bitstream* pour configurer la PRR choisie est de $BS_{size} = 227\,700\ B$. Les mesures ont montré un temps de reconfiguration de $T_{Reconfiguration} = 422\ ms$ identique à quelques millisecondes de différence entre les quatre configurations testées. Ces différences sont probablement causées par l'accès à la CompactFlash. Finalement, le temps pour reconfigurer un mot, T_{word}, est calculé à partir de l'équation suivante :

$$T_{word} = \frac{T_{Reconfiguration}}{BS_{size}} \times 4 \tag{3.4}$$

donc $T_{word} = 7.4\ \mu s$ dans notre cas.

La précision des deux modèles est évaluée par l'erreur moyenne et l'écart type de la puissance par rapport à la mesure. La puissance moyenne de la reconfiguration est directement liée à l'énergie consommée divisée par le temps de reconfiguration. Comme le temps de reconfiguration est le même dans les quatre conditions expérimentales, l'énergie est comparable au même titre que la puissance moyenne pour les cas considérés. Les résultats d'énergie consommée sont présentés dans la table 3.2 avec les estimations par les modèles et les erreurs. La table 3.3 présente l'écart type

d'estimation en puissance.

3.2.3.2 Précision énergétique

Les mesures peuvent être regroupées deux à deux, les Figures 3.7 et 3.8 représentent la consommation pour configurer T_1 alors que T_2 était précédemment configurée et inversement. Les deux tâches présentent des consommations *idle* très proche tandis que les Figures 3.9 et 3.10 présentent la configuration de tâches ayant une consommation *idle* différente, incluant la tâche faible consommation, T_{blank}.

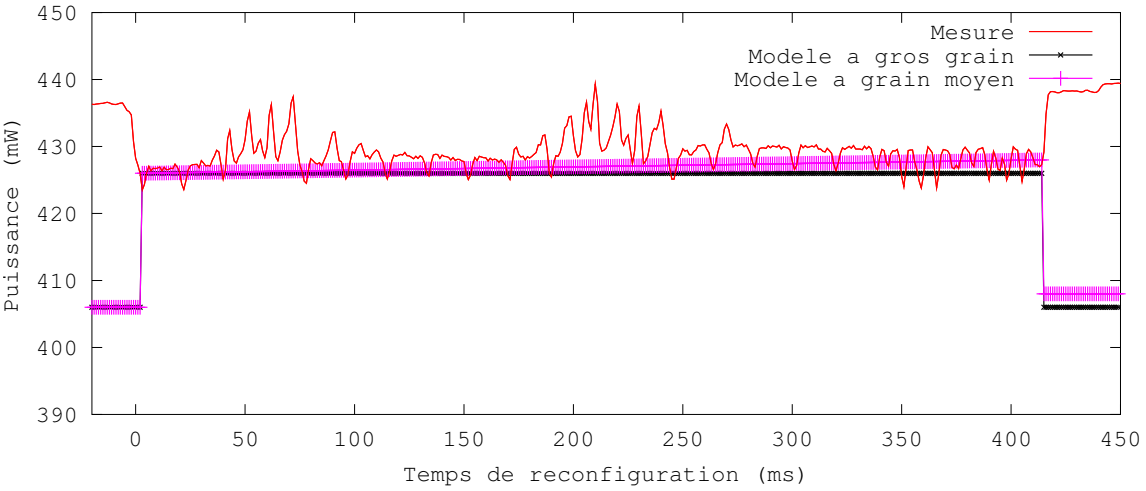

FIGURE 3.7: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_2 vers T_1 en fonction du temps.

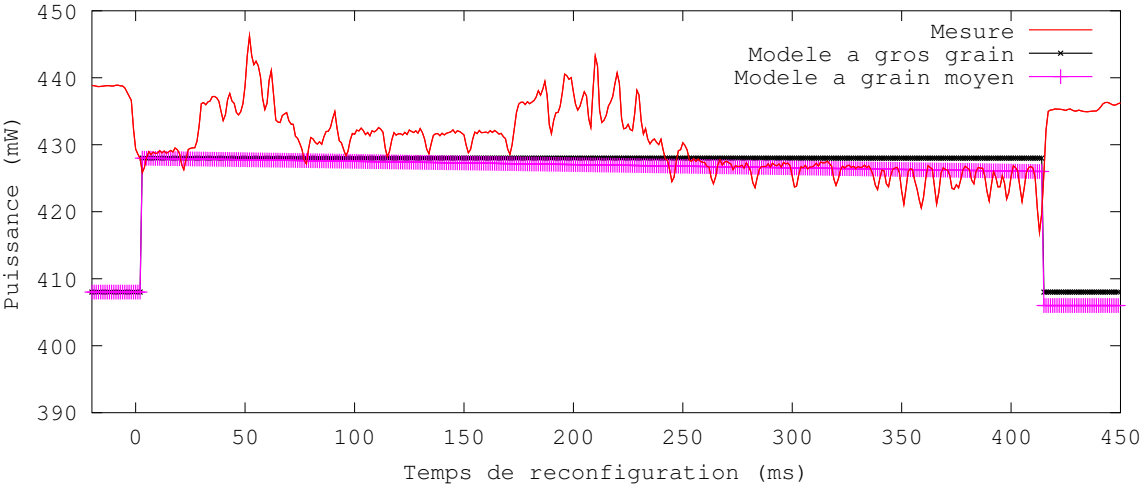

FIGURE 3.8: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_1 vers T_2 en fonction du temps.

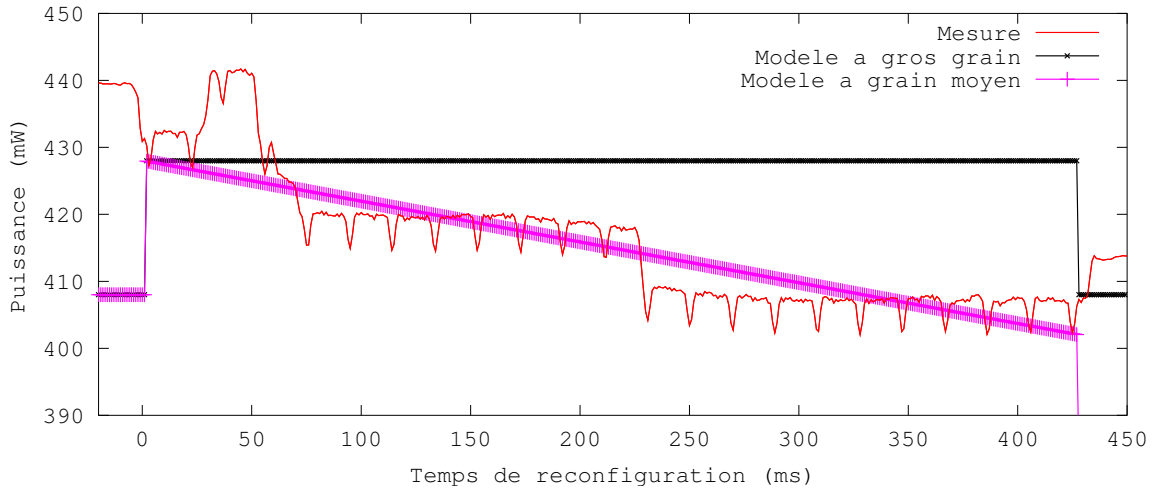

FIGURE 3.9: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_1 vers T_{blank} en fonction du temps.

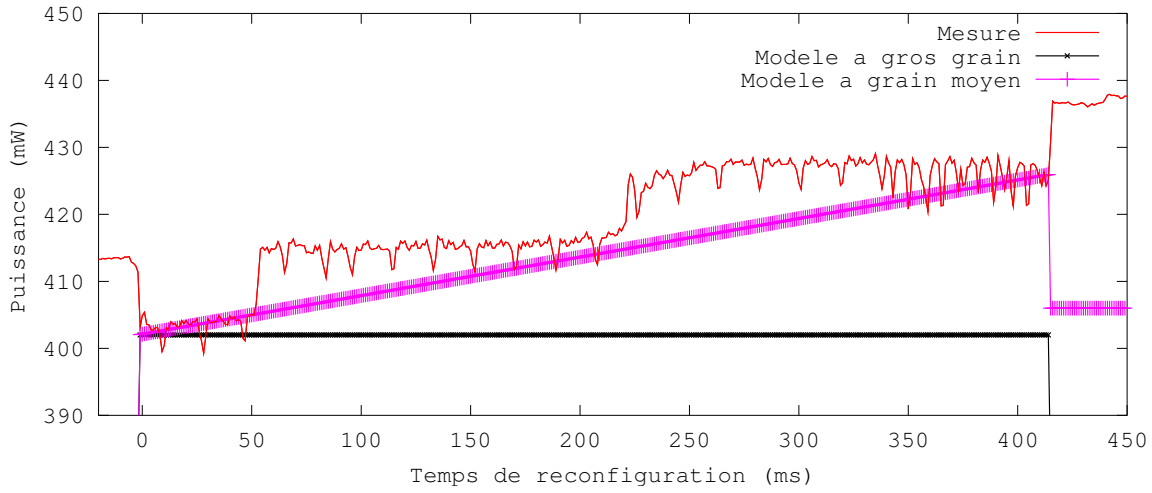

FIGURE 3.10: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_{blank} vers T_2 en fonction du temps.

Dans le premier cas (Figure 3.7 et Figure 3.8), les deux modèles représentent bien la tendance de la puissance consommée par le cœur du FPGA avec une surconsommation de 20 mW liée à l'activité de reconfiguration. Les faibles différences de consommation *idle* (2 mW) entre les tâches présentes avant et après la reconfiguration rendent la différence faible entre les deux modèles. L'énergie est en moyenne bien estimée, l'erreur d'estimation est de 8% et 10% respectivement pour le modèle à gros grain et le modèle à grain moyen. Ceci est normal puisque la puissance $P_{controller}$ a été calculée par moyenne sur les mesures effectuées, la moyenne des résultats de l'estimation lisse les erreurs d'estimation et le résultat est plutôt bon. Cependant

TABLE 3.2: Énergie mesurée (M) pendant la reconfiguration dynamique et erreur énergétique en utilisant le modèle à gros grain (CG, *Coarse Grain*) et à grain moyen (MG, *Medium Grain*).

Config	M (mJ)	erreur CG (mJ)	erreur MG (mJ)
T_2 à T_1	9.12	-1.3 (-13.7%)	-0.84 (-9.2%)
T_1 à T_2	9.58	-0.88 (-9.2%)	-1.3 (-13.5%)
T_1 to T_{blank}	7.97	6.2 (77%)	0.59 (7.4%)
T_{blank} to T_2	10.41	-7.1 (-67.9%)	-2.1 (-19.7%)
Moyenne	9.27	**-0.77 (-8.3%)**	**-0.9 (-9.8%)**
Moyenne absolue	9.27	**3.87 (41.7%)**	**1.2 (13%)**

ces résultats en moyenne cachent des disparités importantes. Dans le second cas (Figure 3.9 et Figure 3.10), le modèle à gros grain ne prend pas en compte les variations de la puissance *idle*, ce qui provoque de fortes différences de consommation, sous estimées ou surestimées selon le cas. L'erreur d'estimation énergétique atteint 77%. Alors que, grâce à la prise en compte de la variation de la consommation *idle*, le modèle à grain moyen limite l'erreur maximale à 19.7%.

3.2.3.3 Précision en puissance

Dans le premier cas (Figure 3.7 et Figure 3.8), malgré la bonne représentation de la tendance de la puissance consommée par le cœur du FPGA liée à l'activité de reconfiguration, on note une sous estimation récurrente de l'énergie par les deux modèles : -8.3% et -9.8%. Ceci est causé par la présence d'une surconsommation, notamment sur la première moitié de la reconfiguration, qui provoque des pics de 10 à 20 mW. Ces pics engendrent un écart type compris entre 2.7 et 4.9 mW pour les deux modèles. Dans le second cas (Figure 3.9 et Figure 3.10), le modèle à gros grain ne prenant toujours pas en compte les variations de la puissance *idle*, les fortes différences de consommation causées par ce phénomène engendrent un écart type de la puissance plus important, entre 8.2 et 10 mW.

La non estimation des pics de puissance ainsi que le non suivi de la variation de la consommation *idle* engendrent un écart type moyen de la puissance estimée de 6.5 mW pour le modèle à gros grain, soit 2.45 mW de plus que le modèle à grain moyen qui intègre une estimation linéaire de la variation de la consommation *idle*.

3.2.4 Conclusion

Ces modèles proposent une estimation de la puissance (et de l'énergie) de la reconfiguration dynamique à deux niveaux de granularité et de complexité. Malgré une erreur d'estimation moyenne de l'énergie assez précise, de l'ordre de 3 à 6%, des

TABLE 3.3: Écart type entre les puissances estimées par les deux modèles comparées à la puissance mesurée.

Config	CG (mW)	MG (mW)
T_2 à T_1	2.7	2.7
T_1 à T_2	4.9	4.6
T_1 à T_{blank}	10	5
T_{blank} à T_2	8.2	3.9
Moyenne	**6.5**	**4.05**

disparités importantes sont constatées selon les configurations de tâches configurées, notamment avec le modèle à gros grain. De plus, la mesure rapporte des pics de consommation et une surconsommation et qui ne sont pas estimés par les modèles, ces pics peuvent poser des problèmes dans certains systèmes avec des contraintes de puissance (vieillissement prématuré des batteries par exemple) et doivent être considérés. L'estimation de la puissance et de l'énergie requise pour la reconfiguration peut nécessiter un modèle à grain très fin pour permettre d'effectuer les bons choix d'implémentation des accélérateurs matériels tout en respectant un budget d'énergie contraint. C'est le cas par exemple dans les systèmes où les temps d'exécution (ou puissance) des tâches accélérées matériellement sont du même ordre de grandeur que les temps (ou puissance) de reconfiguration. Pour ces raisons, nous proposons une analyse plus détaillée de la consommation de la reconfiguration dynamique afin de déterminer quels sont les paramètres faisant varier la puissance. A partir de cette étude, nous proposons dans la suite de ce chapitre, un modèle très fin et sa validation.

3.3 Analyse détaillée de la consommation

Le processus de reconfiguration dynamique consiste principalement à transférer des données entre les éléments en jeu (CompactFlash, mémoire du *MicroBlaze*, contrôleur de reconfiguration et la mémoire de configuration) avant d'obtenir la configuration effective.

La procédure complète peut être découpée par les étapes suivantes, les données de configuration sont :

- chargées depuis le fichier stocké sur la mémoire CompactFlash et mises en tampon dans la mémoire du *MicroBlaze*,
- écrites depuis la mémoire du *MicroBlaze* vers le contrôleur de reconfiguration, *xps_hwicap*,
- écrites depuis la contrôleur de reconfiguration vers le port interne de configuration (ICAP),

- écrites depuis l'ICAP vers la mémoire de configuration,
- appliquées depuis la mémoire de configuration sur les ressources à configurer (CLB, BRAM, DSP, interconnexions).

Dans ces circonstances, la puissance globale consommée par la reconfiguration est le résultat de la combinaison de deux éléments principaux : l'activité liée au transfert et à la gestion du *bitstream* et la configuration réelle des ressources du FPGA. Afin de mieux comprendre l'impact de la configuration des ressources sur la puissance, la sous section suivante propose une analyse du contenu du fichier de configuration.

3.3.1 Organisation du fichier de configuration

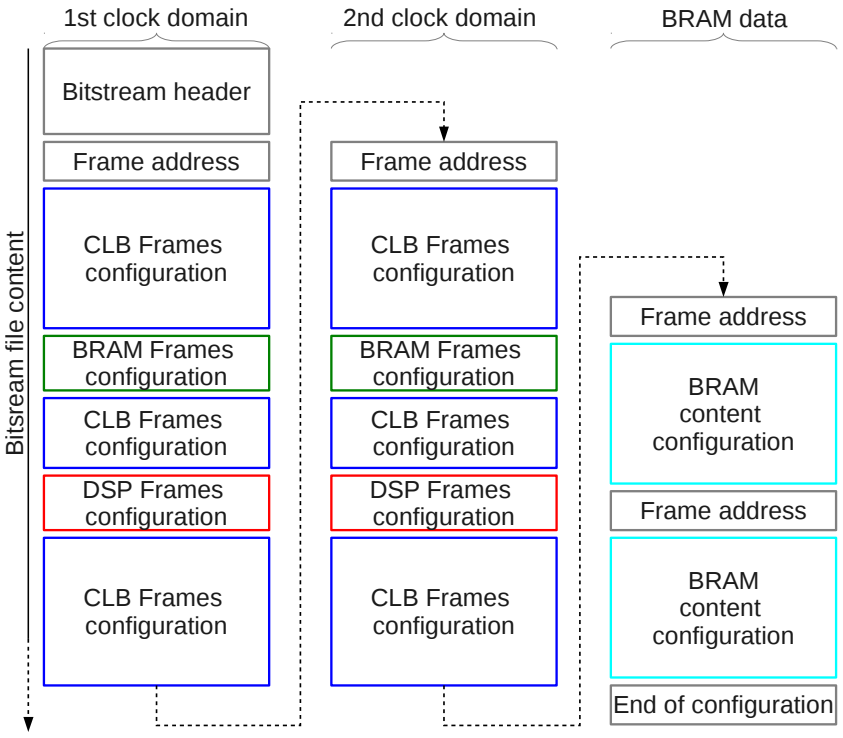

FIGURE 3.11: Composition du *bitstream* pour configurer la PRR utilisée lors de l'expérimentation.

La Figure 3.11 représente l'organisation du fichier de configuration pour configurer la PRR présentée dans la Figure 3.5. Ces informations sont obtenues par analyse du contenu du fichier à partir des informations fournies par Xilinx [84]. Cette PRR s'étend sur deux domaines d'horloges de haut et 16 colonnes de large contenant des CLBs, BRAMs et DSPs. Les premiers mots du fichier de configuration sont un entête qui contient les informations sur le *bitstream*. Le mot suivant contient l'adresse de la première *frame* à configurer. Puis les mots de configuration pour les blocs de CLBs

suivent. Après quatre colonnes de CLBs, la configuration pour une colonne de BRAM est présente. Seule la configuration du BRAM et ses connexions sont présentes, le contenu du BRAM est chargé dans une autre partie du *bitstream*. Une autre adresse de frame correspondant à la première colonne du second domaine d'horloge de la PRR est fournie et la configuration suit la même structure que présentée pour le premier domaine d'horloge. Finalement, le contenu du BRAM est adressé et chargé, puis quelques mots définissent la fin de la reconfiguration.

3.3.2 Lecture et écriture de la configuration

La Figure 3.12 (haut) rapporte le profil de consommation du cœur du FPGA pendant la reconfiguration de T_1 à T_2 (bleu) et de T_2 à T_1 (rouge). Si l'on observe l'ensemble de la courbe, on note clairement qu'il y a des variations de puissance dues à la reconfiguration, telles que décrites lors de la présentation des modèles précédents.

La seconde courbe de la Figure 3.12 est un agrandissement sur les 10 premières millisecondes de la consommation pendant la reconfiguration dynamique. Pour permettre une analyse plus précise, nous avons introduit des marqueurs pilotés par le processeur pour mettre en évidence chaque étape du processus.

Sept étapes sont identifiées et présentées dans le diagramme de séquence, Figure 3.13 :

1) l'arrivée de l'ordre de reconfiguration,

2) l'ouverture du fichier *bitstream* sur la CompactFlash,

3) la lecture de l'entête du *bitstream*,

4) la vérification de la validité de l'entête du *bitstream*,

5) la lecture d'un fragment du fichier sur la CompactFlash,

6) l'écriture des données dans *xps_hwicap*,

7) la répétition des étapes 5 et 6 jusqu'à la fin du fichier.

La consommation en puissance pendant la lecture de la CompactFlash (phases 3 et 5) est faible. À cause de la lenteur de la CompactFlash, le *MicroBlaze* attend les données et génère peu d'activité. Cependant, l'écriture (phase 6) implique plus d'activité, avec le fonctionnement du bus, la gestion de la reconfiguration et la configuration des ressources. La surconsommation correspondante est d'environ 45 mW ce qui représente plus de 10 % de la consommation globale du FPGA. Il y a aussi une autre surconsommation lors de la vérification de la validité du *bitstream* (phase

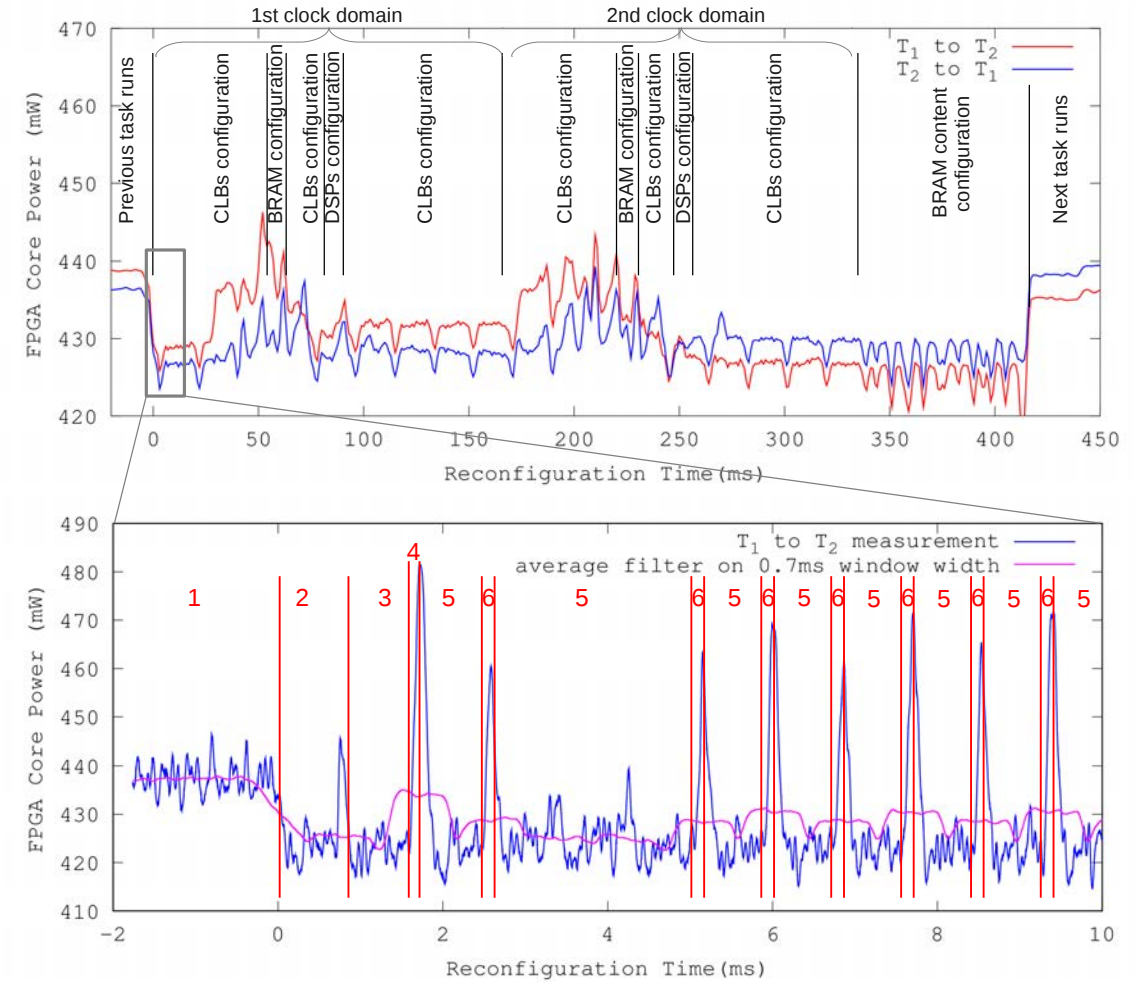

FIGURE 3.12: La courbe du haut montre la consommation en puissance du cœur du FPGA pendant la reconfiguration de T_1 vers T_2 en fonction du temps, en bleu, et de T_2 vers T_1, en rouge. La composition du *bitstream* est mise à l'échelle du temps de reconfiguration pour mettre en évidence les étapes de la reconfiguration. La courbe du bas est un agrandissement temporel du début de la reconfiguration.

4). Cette surconsommation est encore plus importante car elle est provoquée directement par le traitement du processeur *MicroBlaze* qui représente une part de consommation importante. Notons que ces pics de puissance ne sont pas visibles Figure 3.12 à cause d'un filtre moyenneur mis en place pour rendre la courbe plus lisible.

3.3.3 Application de la configuration

Précédemment, le bas de la Figure 3.12 a clairement montré l'influence du transfert des données de configuration sur la puissance. La courbe en magenta montre un moyennage de la consommation à l'aide d'une fenêtre glissante sur 0.7 *ms*. La

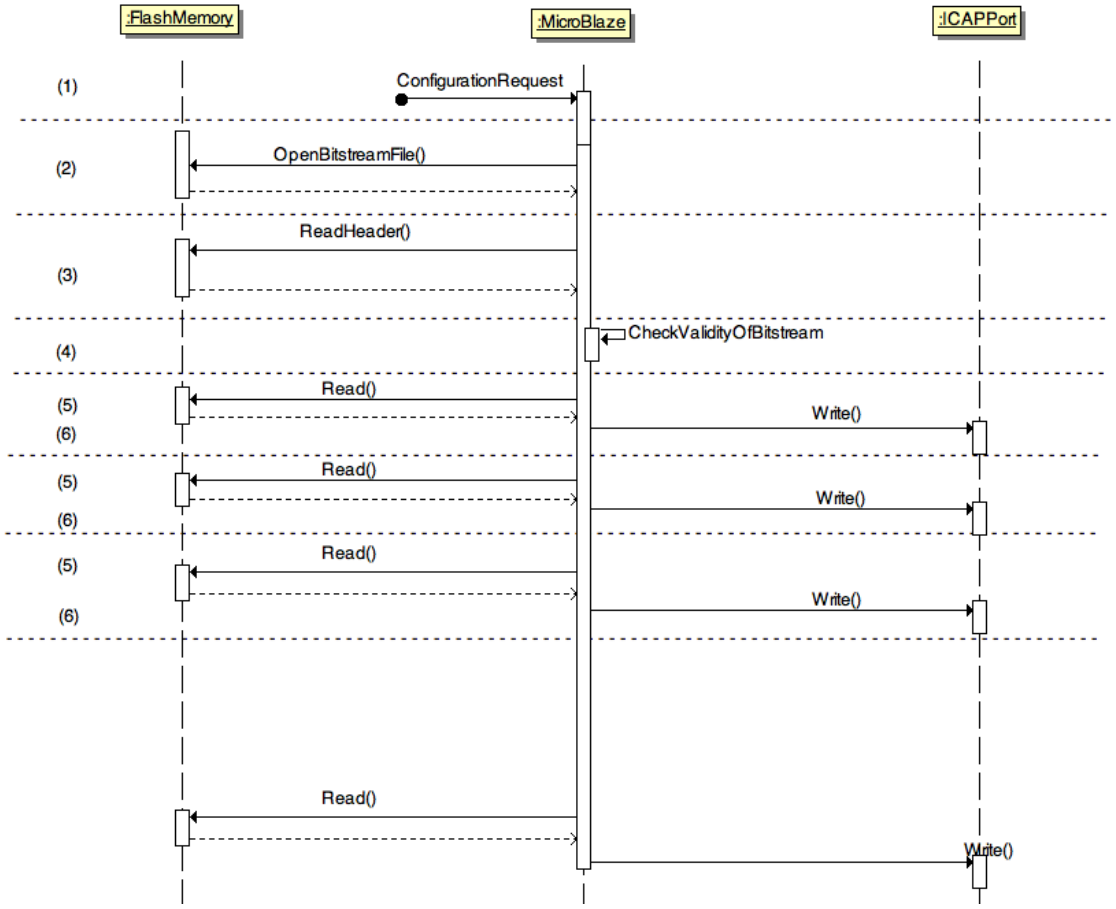

FIGURE 3.13: Diagramme de séquence du processus de reconfiguration.

consommation moyenne sur les sept premiers cycles de lecture et d'écriture est sensiblement au même niveau pour chaque cycle. Ceci devrait être alors le cas, en dehors d'une intervention externe, sur l'ensemble de la reconfiguration. Cependant les deux courbes présentées en haut de la Figure 3.12 (qui ont été filtrées) ne présentent pas une consommation aussi constante que prévue à partir des premiers cycles de lecture/écriture. Premièrement, des surconsommations sont visibles et sont assimilables à deux "vagues" localisées autour de 50 et 200 *ms*. Deuxièmement, la puissance consommée juste au début et juste à la fin de la reconfiguration n'est pas la même. Cette puissance est liée à la consommation *idle* de la tâche en cours de configuration, lorsque celle-ci n'est pas en cours d'exécution comme c'est normalement le cas lors de la reconfiguration.

La forme générale des courbes de puissance, avec les variations constatées, laisse supposer que d'autres paramètres que le contrôleur de la reconfiguration engendrent des variations de la puissance consommée.

Nous faisons l'hypothèse que les variations de consommation sont aussi dues au contenu du *bitstream* et supposons que cette consommation peut être séparée en deux

comportements, des “vagues” de consommation et des “sauts” de consommation, soit les hypothèses suivantes :

- la reconfiguration peut causer des surconsommations transitoires causées par la modification de la configuration et des interconnexions,

- la reconfiguration peut causer des sauts de consommation *idle* par activation/désactivation de blocs.

Les sections suivantes présentent l'analyse de la consommation et nous permettent de confronter nos hypothèses au comportement réel du circuit durant la phase de reconfiguration d'une PRR.

3.3.3.1 Surconsommations transitoires

La Figure 3.12 présente la consommation en puissance durant la reconfiguration de la tâche T_2 alors que T_1 était configurée précédemment et vice versa. Les deux courbes ont la même forme qui est principalement causée par les opérations de modification de la mémoire de configuration. En effet, lorsqu'une tâche est configurée à la place d'une autre tâche, le processus de reconfiguration consiste à écrire les données dans la mémoire de configuration. Si les deux $n^{ièmes}$ mots du *bitstream* des tâches T_1 et T_2 sont les mêmes, dans ce cas la case mémoire contenant ce mot ne change pas et le remplacement du premier mot par le second n'engendre alors qu'une très faible consommation. Inversement, si les deux $n^{ièmes}$ mots des deux *bitstreams* sont exactement complémentaires, dans ce cas chaque bit de la cellule mémoire change lors de l'écriture du second mot en remplacement du premier. Ceci conduit à une augmentation de la consommation. À partir de cette observation, nous supposons qu'une partie de la consommation durant la reconfiguration est liée à la différence entre les *bitstreams* de la tâche implémentée avant et celle implémentée après la reconfiguration. Ces différences peuvent être quantifiées par une métrique comme la distance de Hamming. Cette métrique (initialement utilisée en télécommunication pour obtenir le nombre de bits altérés dans la transmission d'une donnée) indique le nombre de bits différents entre deux mots .

Sur les courbes de la Figure 3.12, deux zones remarquables sont présentes : la première est localisée autour de 50 ms et la seconde autour de 200 ms. L'amplitude de ces surconsommations est d'environ 15 mW dans ce cas. Une analyse approfondie du contenu du *bitstream* montre que ces deux zones correspondent approximativement au moment de la configuration des frames de BRAM. Dans la PRR sélectionnée lors des tests, les BRAMs sont situées sur deux domaines d'horloge ce qui explique la

présence de deux zones de surconsommation. En mettant en relation ces informations avec la vue du placement et routage du projet dans l'outil de placement de Xilinx (PlanAhead), on peut voir que le taux d'occupation des *slices* placées à proximité des frames de BRAM est plus important comparé aux slices qui sont situées plus loin. Ce placement de la logique autour des blocs de BRAMs peut être expliqué par l'algorithme de placement et routage qui essaye probablement de grouper les ressources utilisées dans le but de limiter les coûts des interconnexions vers les mémoires.

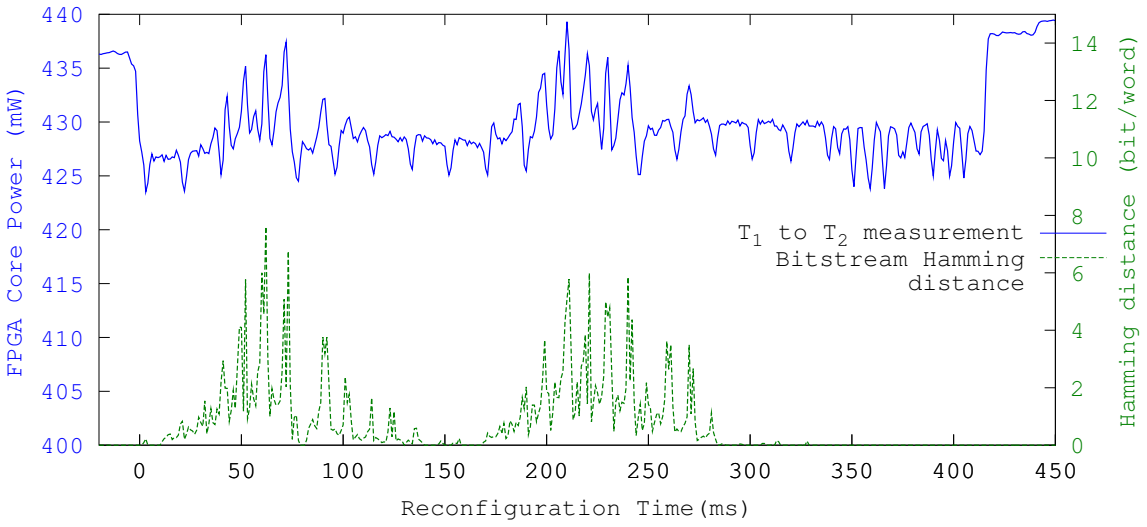

FIGURE 3.14: Puissance consommée par le cœur du FPGA pendant la reconfiguration à partir de T_1 vers T_2 en fonction du temps, en bleu, et la distance de Hamming par mot de configuration entre les *bitstreams* de T_1 et T_2 en fonction du temps de reconfiguration, représenté par la ligne verte en pointillés.

Le haut de la Figure 3.14 représente la consommation en puissance du cœur du FPGA pendant la reconfiguration de la tâche T_2 sur une PRR où une tâche T_1 était configurée précédemment. Le bas de cette figure représente la distance de Hamming (calculée mot par mot de 32 *bits*) entre les *bitstreams* des deux tâches concernées. Les abscisses ont été converties pour correspondre au temps de reconfiguration (ms). On remarque que la forme de la distance de Hamming entre le *bitstream* de la tâche précédemment implémentée et celui de la nouvelle tâche est très proche du profil de consommation. La distance de Hamming montre des pics aux instants où les surconsommations sont présentes, autour de 50 ms et $200 - 250$ ms.

Cette concordance consolide l'hypothèse d'un lien entre les différences de configuration et les surconsommations.

Dans [70] les auteurs précisent que le changement de configuration d'un composant configurable ATMEL provoque des courts-circuits au niveau des interconnexions. Cette information suggère que les surconsommations résultent de l'activation et désactivation des interconnexions entre les ressources du FPGA. Ceci peut causer des connexions non voulues et peut être des petits courts-circuits alors que la PRR n'est pas complètement configurée. Les modifications des interconnexions dans le FPGA sont précisées par le *bitstream* et le nombre d'interconnexions modifiées se reflète dans la distance de Hamming des mots de configuration. À ce stade et avec cette plateforme, il est difficile de déterminer quels sont les bits du *bitstream* qui codent les interconnexions et qui devraient être pris en compte pour améliorer les modèles précédents.

3.3.3.2 Variations de la consommation *idle*

En analysant plus finement la Figure 3.12, on peut observer que les niveaux de puissance au début et à la fin de la reconfiguration sont différents et sont liés aux tâches concernées par la procédure de reconfiguration.

Cette différence est concordante avec la consommation des tâches présentée dans la table 3.1. En effet, les deux tâches T_1 et T_2 n'ont pas le même nombre de ressources utilisées et n'ont pas la même puissance *idle*. Cette puissance *idle* est liée à la taille des tâches et au taux d'occupation des ressources, ainsi une même tâche implémentée sur deux PRRs différentes aura la même consommation *idle*. L'analyse de la consommation pendant la reconfiguration d'une tâche ayant une grande différence de puissance *idle* par rapport à la tâche précédemment configurée permettra de mettre en évidence le comportement de la variation de la puissance *idle*.

Pour illustration, la Figure 3.15 représente la consommation du cœur du FPGA pendant la reconfiguration dynamique de la PRR de la tâche T_{blank} vers T_1, où T_{blank} est une tâche vide. Cette figure montre explicitement deux sauts de puissance pendant la reconfiguration. Ces sauts amènent la puissance du niveau de la tâche présente avant la reconfiguration, T_{blank}, vers la consommation *idle* de la tâche nouvellement configurée, T_1.

L'analyse avancée de la décomposition du *bitstream*, présentée Figure 3.15, montre qu'un saut de consommation apparaît juste avant la configuration des interconnexions des BRAMs sur le FPGA utilisé pour l'expérience (Xilinx Virtex-5). Ce comportement est typique de l'activation ou de la désactivation d'éléments. Les BRAMs sont connus pour avoir une consommation non négligeable, l'estimateur proposé par Xilinx [39] donne une consommation d'une BRAM (pour Virtex 6) pour environ 1.67 mW même si aucun transfert n'a lieu lorsque les signaux d'activation (*enable*) des ports sont activés. Les tâches T_1 et T_2 utilisées ont la même quantité de

BRAMs requise mais la quantité de slices et de DSPs n'est pas la même. Il probable que la distribution de l'horloge sur les *slices* (CLBs) soit activée ou désactivée ainsi que l'activation des blocs DSPs en fonction des besoins en ressource de chaque tâche. Selon cette hypothèse, il y a un lien direct entre le taux d'utilisation des ressources (donc entre la configuration de la PRR considérée) et la consommation *idle* de la PRR. Ceci justifie la nécessité de prendre en compte ces variations de consommation *idle* dans le modèle de la reconfiguration dynamique.

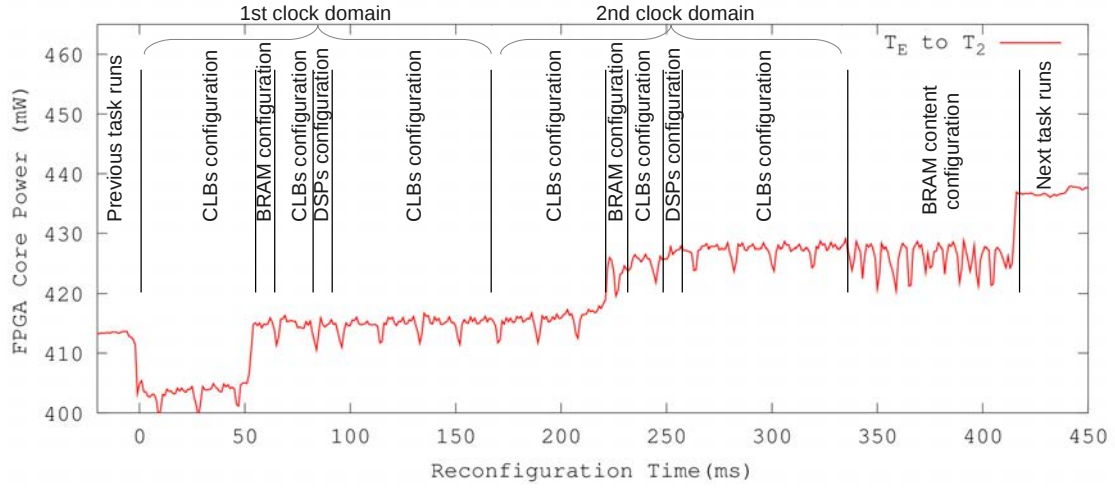

FIGURE 3.15: Puissance consommée par le cœur du FPGA pendant la reconfiguration dynamique d'une PRR contenant T_{blank} vers la configuration de T_2 en fonction du temps. La composition du *bitstream* ramenée à la même échelle de temps est aussi présentée.

Cette section a présenté l'analyse de la puissance consommée durant la reconfiguration dynamique. Nous avons montré qu'il y a trois composantes principales qui participent à la consommation de la reconfiguration dynamique et confirmé les hypothèses de départ. Premièrement il y a un surcoût dû au contrôle de reconfiguration pour transférer le *bitstream*. Ensuite il y a des surconsommations causées par les différences entre la précédente et la nouvelle configuration en particulier au niveau des interconnexions. Finalement il y a des variations brusques de consommation *idle* liées à la mise en place des ressources de la nouvelle tâche configurée. À partir de ces mesures détaillées et de cette analyse, la section suivante propose la construction d'un modèle à grain fin pour estimer précisément les variations de la consommation pendant la reconfiguration.

3.3.4 Extraction du modèle à grain fin

Dans cette section, nous présentons un modèle plus fin de la puissance de la reconfiguration basé sur les constatations présentées dans la section précédente.

Contrairement au modèle à grain moyen basé sur une interpolation où le profil de puissance varie linéairement de P^{idle}_{prev} à P^{idle}_{next}, ici, ce profil évolue non linéairement. Les variations de consommation sont prises en compte dans cette estimation grâce à une analyse du contenu du *bitstream* et du calcul de la distance de Hamming entre les deux *bitstreams* des tâches précédemment configurée et nouvellement configurée.

L'introduction d'une nouvelle fonction, $steps(t)$, permet de suivre les variations de la consommation *idle* lors de la reconfiguration en fonction des paramètres physiques du FPGA. Le résultat de cette fonction, compris entre 0 et 1, dépend de la technologie, de la taille et position des ressources dans la PRR. Dans nos conditions expérimentales (Virtex 5), la valeur de cette fonction est liée à la position des BRAMs. Le résultat de cette fonction est calculé à partir de la vitesse de reconfiguration, du contenu et de la taille du *bitstream*. Plusieurs valeurs intermédiaires (comme $0; 0.5; 1$ dans notre cas) sont possibles en fonction de la taille et contenu de la PRR. Cette fonction est obtenue par caractérisation en puissance de la reconfiguration du FPGA. L'analyse porte ici sur le Virtex-5, mais le Virtex-6 a la même architecture et sa consommation a un comportement similaire. Le détail du calcul de la fonction est présenté par la suite.

La fonction de calcul de la distance de Hamming entre la configuration précédente et la suivante de la PRR est introduite également, $d_{Hamming}(\tau, T_{prev}, T_{next})$. La distance de Hamming est calculée sur des mots de 32 *bits* avec une moyenne par fenêtre flottante de 100 mots entre la configuration de la tâche précédemment configurée et la suivante. Cette moyenne est requise car des différences à différents mots d'une *frame* n'engendrent pas les mêmes variations de puissance en fonction de leur rôle dans la configuration et ce niveau de détail n'est pas connu. La fenêtre de moyennage est adaptée dans ce cas pour couvrir une *frame* complète, donc au minimum 82 mots [84] que nous avons choisi d'arrondir à 100. Le moyennage de la distance de Hamming permet une estimation de la consommation pendant la reconfiguration prenant en compte les modifications dans une *frame* complète, au plus proche de l'impact sur la consommation.

Le modèle suit l'équation suivante :

$$\begin{aligned} P_{FG}(t) &= P^{idle}_{FPGA} + P^{idle}_{prev} + P_{controller} \\ &+ steps(t) \times (P^{idle}_{prev} - P^{idle}_{next}) \\ &+ \alpha \times d_{Hamming}(t, T_{prev}, T_{next}) \end{aligned} \tag{3.5}$$

avec α le coefficient dépendant de la technologie du circuit. Il est calibré en utilisant un algorithme itératif de minimisation de l'erreur moyenne entre l'estimation du modèle et une mesure de consommation lors de la reconfiguration. Ce paramètre correspond au poids de la distance de Hamming dans le modèle et dépend de la façon dont sont configurées les *frames*.

La Figure 3.16 présente un profil de consommation représentant le modèle exposé dans l'équation 3.5. Sur cette figure on distingue les trois contributions en puissance modélisées dans le modèle à grain fin : la puissance *idle* de la PRR (en vert) avec la variation en échelons, la puissance dynamique du contrôleur de reconfiguration (en jaune) et la puissance due aux différences du contenu des *bitstreams* (en orange), au dessus de la consommation *idle* de base du FPGA, constante durant l'opération (en vert). Dans les conditions d'expérimentations, P^{idle}_{FPGA} vaut toujours 402 mW, de même pour $P_{controller} = 20\ mW$. P^{idle}_{prev} et P^{idle}_{next} sont choisies à partir de la table 3.1 en fonction du scénario de configuration.

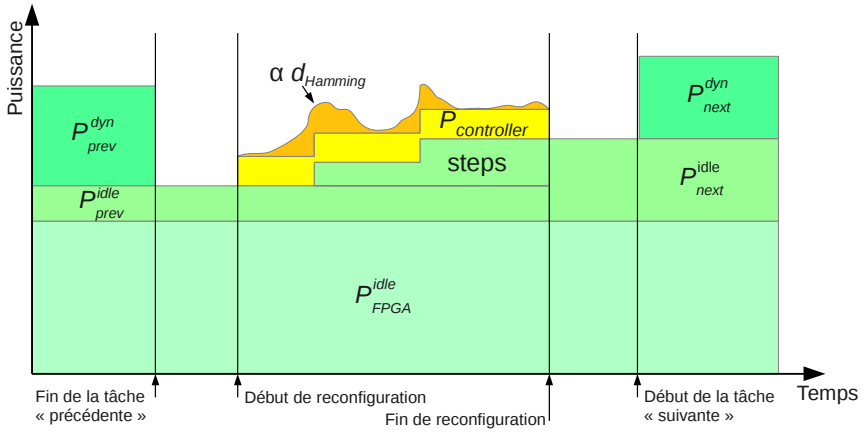

FIGURE 3.16: Représentation du profil de consommation de la reconfiguration dynamique en tenant compte de la consommation des variations de consommation selon le contenu de la PRR et la distance de Hamming, selon l'équation 3.5.

En considérant le cas de la reconfiguration de T_2 vers T_1, nous avons trouvé que $\alpha = 0.003$, obtenu pour le Virtex-5 à partir de la mesure du scénario de configuration

T_2 vers T_1, est le facteur de puissance optimal pour la distance de Hamming.

Finalement, $steps(t)$ est obtenu à partir de l'architecture du FPGA tel que détaillé ci-après. Présentée dans la section 3.2.3.1, la PRR considérée est sur deux domaines d'horloges avec chacun une colonne de BRAM. Comme vu dans la section 3.3.3.2, les sauts de consommation interviennent juste avant la configuration des interconnexions des BRAMs. La PRR contient une colonne de BRAM par domaine d'horloge. Puisque la PRR est située sur deux domaines d'horloge, il y a configuration de deux colonnes de BRAM. Ainsi la variable *step* prend trois valeurs ; une première valeur 0 avant la configuration au niveau de la BRAM, puis une seconde valeur 0.5 après la première colonne de BRAM et avant la seconde et finalement, la dernière valeur 1 jusqu'à la fin de la reconfiguration. Ces valeurs $(0; 0.5; 1)$ sont déterminées par proportion égales en fonction du nombre de colonnes de BRAM. Dans le *bitstream*, avant d'atteindre la configuration de la première BRAM, quatre colonnes de CLBs sont présentes. D'après la documentation de Xilinx [84], une colonne de CLB nécessite 36 *frames* pour sa configuration et une *frame* contient 41 mots de configuration. Donc le premier saut apparaît à $41 \times 36 \times 4 = 5\,904$ mots. Et le second saut est à la même position dans le second domaine d'horloge, soit 31 898 mots, sans considérer les commandes spéciales présentes dans le *bitstream* qui contiennent des adressages et NOPs et ne représentent que quelques mots. Alors

$steps(t) = 0 \quad \forall t \in [0 - 5\,903 \times T_{1word}]$,
$steps(t) = 0.5 \quad \forall t \in [5\,904 \times T_{1word} - 31\,897 \times T_{1word}]$,
$steps(t) = 1 \quad \forall t \in [31\,898 \times T_{1word} - 56\,925 \times T_{1word}]$

correspondent aux positions des variations brusques de la consommation *idle* lors de la reconfiguration.

Ce modèle est appelé modèle à grain fin du fait de la finesse des paramètres utilisés, notamment le calcul de la distance de Hamming. Ces paramètres et fonctions, issus de l'analyse détaillée des courbes de consommation, calculés pour la plateforme sont appliqués au sein du modèle à grain fin dans la section suivante. Cette section compare et discute de la précision de l'estimation de ce modèle par rapport aux mesures et aux deux modèles présentés au début du chapitre.

3.3.5 Étude de la précision du modèle à grain fin

Les profils de puissance estimés par les trois modèles sont présentés dans les quatre cas de reconfiguration sélectionnés, figures 3.17, 3.18, 3.19 et 3.20.

Les figures montrent visuellement que le modèle à grain fin apporte une précision supplémentaire par rapport aux deux premiers modèles. Similairement aux deux premiers modèles, la précision des estimations en énergie et en puissance sont respectivement présentées dans la table 3.4 et la table 3.5 pour permettre la comparaison

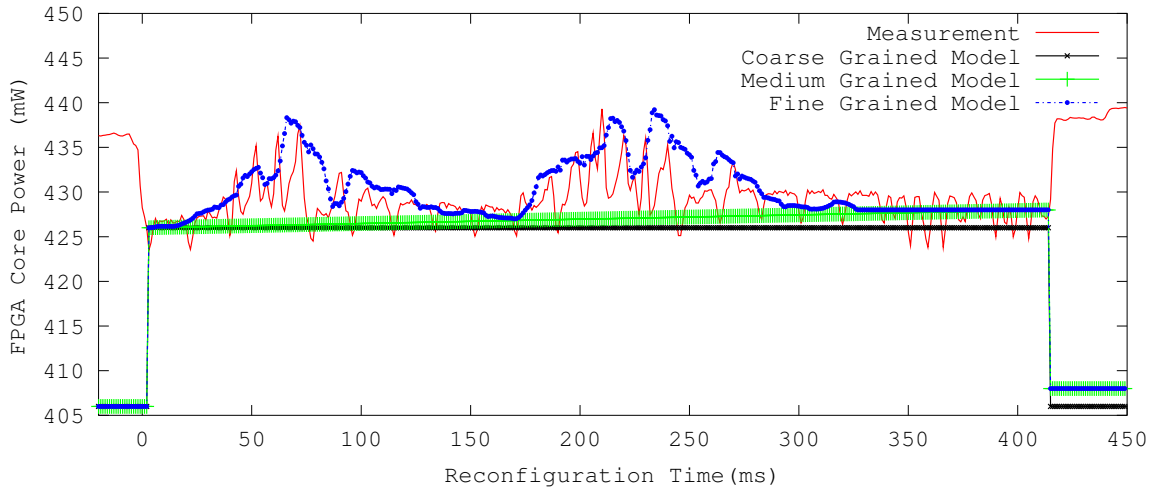

FIGURE 3.17: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_2 vers T_1 en fonction du temps.

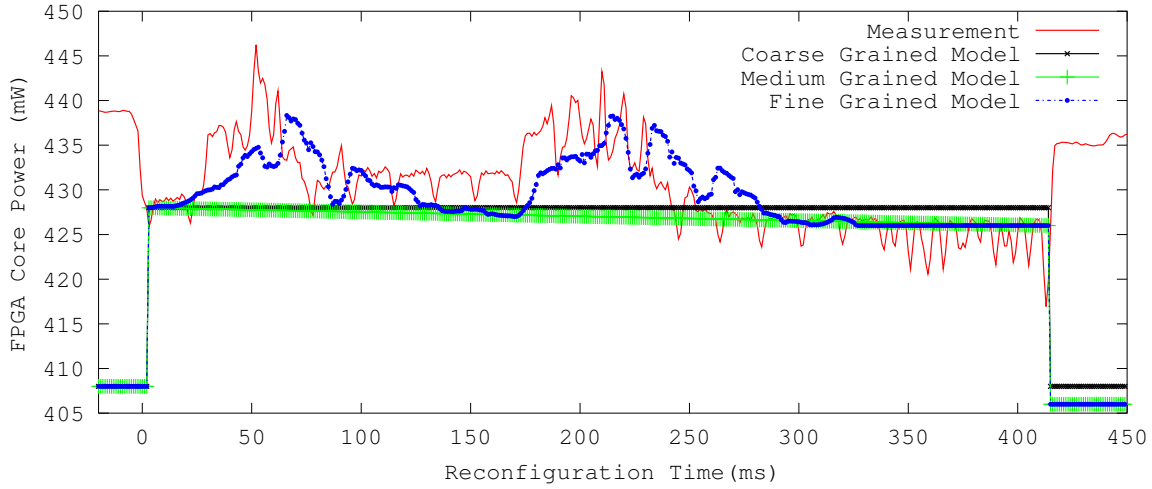

FIGURE 3.18: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_1 vers T_2 en fonction du temps.

objective des trois modèles.

3.3.5.1 Précision énergétique

Au niveau de l'énergie estimée, le modèle à grain fin apporte une nette diminution de l'erreur en moyenne, seulement 1.8% d'erreur, soit plus de 5 fois mieux que le modèle à grain moyen. Le second point important caché par cette moyenne est la disparité de l'erreur en fonction des configurations étudiées. Pour le modèle à grain fin, l'erreur maximale est de 5.5% alors qu'elle est de 19.7% pour le modèle à grain moyen et de 77% pour le modèle à gros grain.

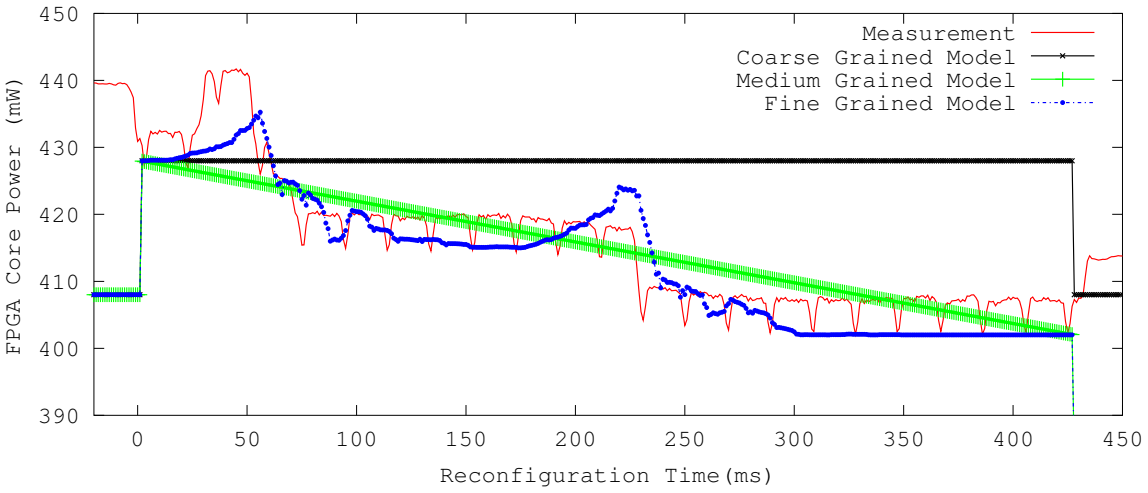

FIGURE 3.19: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_1 vers T_{blank} en fonction du temps.

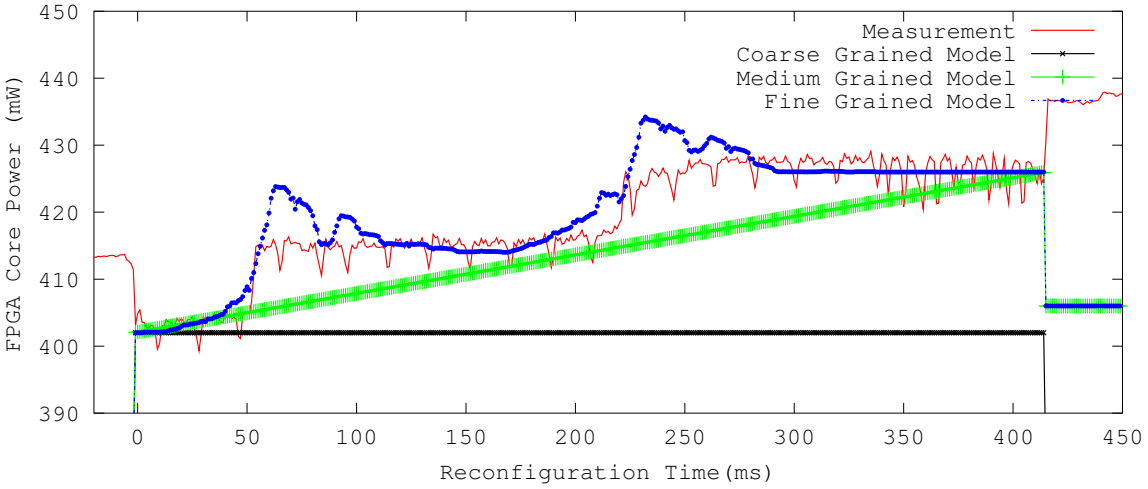

FIGURE 3.20: Mesure et résultats de l'estimation pour la puissance du cœur du FPGA pendant la reconfiguration dynamique de T_{blank} vers T_2 en fonction du temps.

Le modèle à grain fin apporte une meilleure précision de l'énergie consommée par la reconfiguration, ceci dans tous les scénarios de reconfiguration présentés comparé aux modèles à gros grain et à grain moyen.

3.3.5.2 Précision en puissance

Globalement la forme de la courbe est proche de la mesure. Cependant, on peut noter que la courbe estimée ne suit pas forcément la mesure, notamment Figure 3.19 et Figure 3.20 où il y a dans un cas une sous estimation du pic de puissance et dans le second cas une surestimation du pic de puissance. Ces pics de puissance sont

TABLE 3.4: Énergie mesurée (M) pendant la reconfiguration dynamique et l'erreur énergétique en utilisant le modèle à gros grain (CG), à grain moyen (MG) et à grain fin (FG, *Fine Grain*).

Config	M (mJ)	CG error (mJ)	MG err. (mJ)	FG err. (mJ)
T_2 to T_1	9.12	-1.3 (-13.7%)	-0.84 (-9.2%)	0.5 (5.5%)
T_1 to T_2	9.58	-0.89 (-9.2%)	-1.3 (-13.5%)	-0.22 (-2.3%)
T_1 to T_{blank}	7.97	6.2 (77%)	0.59 (7.4%)	-0.13 (-1.7%)
T_{blank} to T_2	10.41	-7.1 (-67.9%)	-2.1 (-19.7%)	0.5 (5%)
Moyenne	9.27	**-0.77 (-8.3%)**	**-0.9 (-9.8%)**	**0.16 (1.8%)**
Moyenne absolue	9.27	**3.87 (41.7%)**	**1.2 (13%)**	**0.34 (3.6%)**

TABLE 3.5: Écart type entre la puissance estimée par les modèles proposés comparée à la puissance mesurée.

Config	CG (mW)	MG (mW)	FG (mW)
T_2 à T_1	2.7	2.7	3.2
T_1 à T_2	4.9	4.6	3.8
T_1 à T_{blank}	10	5	4.3
T_{blank} à T_2	8.2	3.9	3.7
Moyenne	**6.5**	**4.05**	**3.75**

modélisés par la distance de Hamming au niveau du modèle. Ceci signifie que la distance de Hamming, calculée sur l'ensemble du *bitstream*, n'est peut être pas le paramètre optimal pour estimer ces pics. Ces pics de puissance sont potentiellement constitués d'une puissance statique via les configurations des interconnexions. Pour estimer plus précisément les pics de puissance, il faut probablement sélectionner certains bits dans les *frames* et seulement certaines combinaisons correspondant à la configuration des interconnexions. Des informations aussi détaillées ne sont pas fournies par le constructeur, Xilinx, et les expérimentations à mettre en œuvre pour les obtenir sont multiples avec des manipulations avancées du *bitstream* et ne font pas partie de ce travail. Cependant, les chiffres donnés par l'écart type de la puissance montrent une amélioration en moyenne avec le modèle à grain fin. Cette amélioration est masquée par un décalage du modèle par rapport aux pics de puissance réels. Les pics de puissance sont globalement bien estimés, mais leur décalage provoque une augmentation de l'écart type.

3.4 Conclusion

Les trois modèles proposés sont intéressants pour la modélisation de la puissance et de l'énergie des composants reconfigurables à trois niveaux de précision et complexité différents. Il peuvent permettre de cibler différents objectifs.

- Le modèle à gros grain convient bien pour une estimation rapide. Sa bonne précision, en moyenne, pour l'énergie le rend adapté à une estimation rapide d'un système reconfigurable global, à haut niveau.

- Le modèle à grain moyen propose aussi une bonne précision énergétique qui est raisonnable dans tous les cas de reconfiguration et non seulement en moyenne comme le modèle à gros grain. Ce modèle est approprié pour une bonne estimation de la puissance et de l'énergie de chaque procédure de reconfiguration et peut être utilisé pour une estimation énergétique à un niveau plus bas que le modèle à gros grain lorsque les ressources utilisées par les tâches sont précisément connues et que la consommation *idle* peut être précisée.

- Le modèle à grain fin permet d'obtenir une précision en énergie et en profil de puissance. Ce modèle est utilisé dans deux cas, soit pour estimer très précisément la consommation énergétique de la reconfiguration, soit pour l'obtention de courbes de profil de puissance précis.

Le choix du modèle pour l'estimation de la consommation de la reconfiguration dépend aussi des caractéristiques des tâches configurées, plus le temps d'exécution des tâches se rapproche du temps de reconfiguration, plus la reconfiguration a un impact sur la puissance, l'énergie et les performances de l'application. Dans ce cas, l'utilisation du modèle à grain moyen ou grain fin se justifie dans une estimation ou exploration à haut niveau, par exemple pour valider les choix d'allocation pour la faible consommation.

Il est nécessaire de préciser que l'efficacité du contrôleur de reconfiguration a un impact très important sur la puissance globale de la reconfiguration dynamique partielle. Dans les expériences proposées, (basées sur le projet de référence de Xilinx), la vitesse de reconfiguration est contrainte par l'utilisation de la CompactFlash avec *xps_hwicap* et le *MicroBlaze*. Ce choix a été effectué car il s'agit d'un projet de référence du constructeur pour la reconfiguration dynamique et les vitesses de reconfiguration obtenues permettent une analyse détaillée de la puissance instantanée. Des travaux récents, [66] notamment, proposent un processus plus efficace basé sur un contrôleur de reconfiguration où le *MicroBlaze* n'est pas nécessaire et où une mémoire DDR est utilisée à la place de la CompactFlash pour améliorer le débit. Un contrôleur de mémoire externe avec un accès direct permet de limiter l'utilisation d'un processeur dans le transfert des données et limiter la puissance consommée et les délais de transfert. De plus la conception d'un contrôleur spécifique permet de réduire les chemins critiques et d'augmenter la fréquence de fonctionnement. Ces aspects sont détaillés dans le chapitre suivant. Toutefois, une reconfiguration plus

rapide et plus efficace avec un contrôleur différent engendrera les mêmes phénomènes que ceux étudiés dans ce chapitre. Les temps de reconfiguration et la puissance liée au contrôleur seront modifiés. L'utilisation d'un contrôleur de reconfiguration optimisé, permettant de réduire également la puissance consommée par celui-ci, rend proportionnellement plus importantes les variations causées par la reconfiguration et implique qu'elles doivent toujours être considérées pour une estimation en puissance ou énergie précise. L'intérêt d'un modèle à grain fin se justifie également dans ce cas puisque la puissance moyenne du contrôleur devient plus faible par rapport aux autres phénomènes de consommation considérés dans le modèle à grain fin.

4 Exploration de la consommation dans les plateformes reconfigurables

Contenu

Les architectures reconfigurables peuvent offrir des performances élevées et permettent de supplanter l'exécution sur processeur pour beaucoup d'applications. La reconfiguration dynamique permet d'ajouter la dynamique et la variabilité qui manquent aux architectures matérielles. De plus, la reconfiguration dynamique peut être utilisée pour diminuer la consommation énergétique, dans certains cas, en permettant de réduire la surface nécessaire pour une application ou en réduisant la consommation des régions non utilisées par effacement de la configuration.

L'étude des différents paramètres de la consommation de la reconfiguration dans le chapitre précédent est générique pour l'ensemble des contrôleurs de reconfiguration. Cependant le contrôleur pris pour cette étude est lent à l'échelle d'une application (422 *ms* dans le cas présenté). Dans le cas d'un décodeur vidéo, par exemple, les images doivent être traitées au rythme de 25 images par seconde (fps - *Frames Per Second*), soit un temps de décodage complet pour chaque image de 40 *ms*. Cette contrainte de temps ne permet pas d'utiliser la reconfiguration dynamique avec les performances présentées précédemment pour ce type d'application. De plus, la reconfiguration dynamique présentée sollicite le processeur *MicroBlaze* pour le transfert des données ce qui rend à la fois la zone en cours de configuration et le processeur inutilisables.

La mise en place d'un processus de reconfiguration plus performant permettrait

de rendre la reconfiguration dynamique attrayante pour des applications avec des contraintes temporelles et énergétiques fortes. Le chapitre 2 a montré qu'il était possible d'exploiter la reconfiguration dynamique pour mieux respecter ces contraintes, par exemple en adaptant le niveau de parallélisme des tâches matérielles ou par l'effacement de configurations précédentes. Cependant, dans le cadre d'un système complet il est particulièrement difficile d'appréhender les bénéfices de la reconfiguration, surtout concernant la consommation énergétique. En fonction des implémentations déjà effectuées, des exécutions à venir et des contraintes globales du système, un point clé réside dans l'évaluation de l'impact de différents choix de reconfiguration sur la consommation.

Pour répondre aux deux points évoqués et dans le but d'optimiser l'énergie, par exemple, ce chapitre présente d'abord un contrôleur de reconfiguration optimisé, très performant, qui améliore l'applicabilité de la reconfiguration dynamique partielle au sein d'une application. Un algorithme d'exploration des solutions d'implémentation qui minimisent la consommation d'énergie ou le temps d'exécution est ensuite présenté. L'impact de la reconfiguration dynamique, en particulier pour la réduction de l'énergie consommée, est discuté sur un exemple en dernière partie.

4.1 Optimisation du contrôleur de reconfiguration

Les architectures reconfigurables dynamiquement sont utilisées dans différents domaines car elles ont des performances élevées proches des accélérateurs dédiés et une consommation d'énergie plus faible qu'un processeur tout en apportant une flexibilité de configuration. Cependant pour être utilisées dans des systèmes avec des contraintes de temps et d'énergie importantes, la reconfiguration doit être la plus rapide possible tout en maîtrisant la consommation. Cette section présente un contrôleur de reconfiguration optimisé qui permet d'atteindre des débits de reconfiguration jusqu'à 1.433 GB/s et dont les chemins de données sont étudiés pour réduire la consommation.

Ces travaux d'optimisation du contrôleur de reconfiguration ont été effectués avec la contribution de Hung-Manh Pham et Sébastien Pillement en dehors du cadre de cette thèse. Hung-Manh Pham a par la suite travaillé sur l'intégration d'une gestion de la mémoire DDR2/DDR3 à ce contrôleur de reconfiguration [85].

4.1.1 Contrôleur de reconfiguration

La Figure 4.1 détaille la structure interne du contrôleur de reconfiguration proposé, UPaRC (*Ultra-Fast Power-aware Reconfiguration Controller*). Il est constitué d'un contrôleur de reconfiguration rapide, UReC (*Ultra-fast Reconfiguration Controller*)

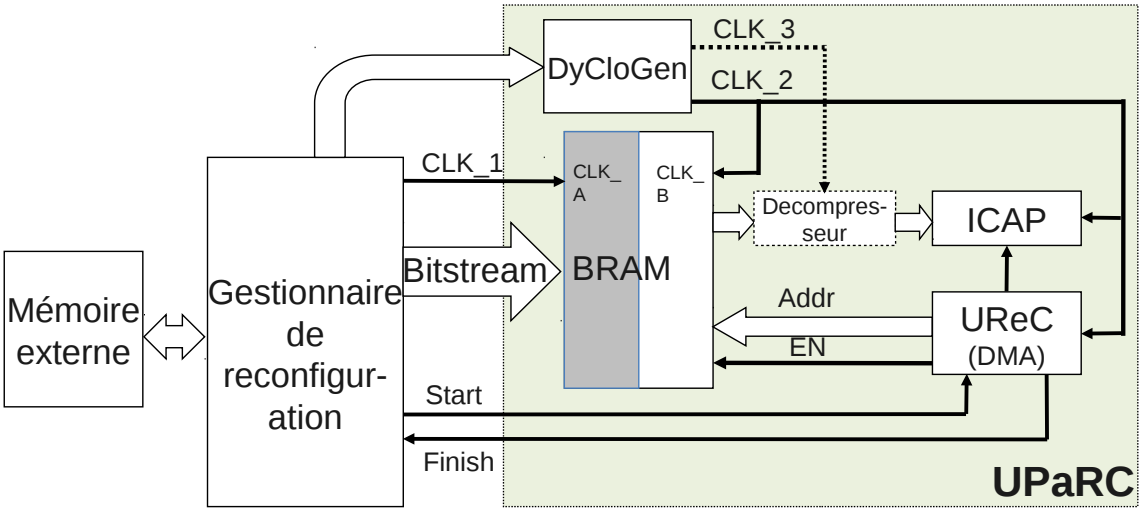

FIGURE 4.1: Structure détaillée de UPaRC.

d'un générateur d'horloge avec variation dynamique de la fréquence, DyCloGen et d'un module pour la décompression de *bitstream*. Le gestionnaire de la reconfiguration (*Manager*), connecté à une mémoire externe, pilote UPaRC pour lancer un processus de reconfiguration rapide.

Le contrôleur de reconfiguration contient une mémoire BRAM pour stocker les bitstreams qui sont utilisés pour la reconfiguration. Une mémoire BRAM à double port est utilisée. Un port est interfacé avec le gestionnaire pour précharger le bitstream tandis que le second port est connecté au bloc UReC. Le préchargement du bitstream (dans la BRAM) n'affecte pas le fonctionnement de la PRR, donc le temps de reconfiguration d'une PRR est uniquement le temps nécessaire pour charger les données de configuration depuis la BRAM vers l'ICAP.

Le bloc UReC contient un accès direct (DMA) au second port de la mémoire BRAM ce qui permet d'optimiser les transferts depuis la BRAM vers l'ICAP. Ceci permet d'atteindre la vitesse maximale de reconfiguration. Cette méthode est aussi utilisée dans d'autres contrôleurs de reconfiguration présentés dans l'état de l'art, cependant ces contrôleurs utilisent un bloc DMA existant dont la taille est importante et dont la fréquence est limitée à 200 MHz. L'interface avec la BRAM est optimisée pour permettre des transferts à fréquence élevée avec un mot de configuration (32 bits) transféré à chaque cycle d'horloge. De plus, le bloc UReC n'effectue que des accès en lecture à la BRAM ce qui permet de réduire au maximum les ressources matérielles nécessaires. Grâce à cela, le bloc UReC occupe une petite surface qui permet de réduire les chemins critiques et d'augmenter les vitesses de transfert. Ce module permet d'utiliser l'ICAP à une fréquence très haute (jusqu'à 362.5 MHz), plus élevée que les caractéristiques données pour le fonctionnement de la BRAM (300 MHz)[86]. C'est pourquoi la fréquence de fonctionnement du bloc UReC est

beaucoup plus élevée que le meilleur contrôleur de reconfiguration de l'état de l'art, FaRM (200 MHz).

Lorsque le bloc UReC reçoit le signal "Start", il sélectionne alors la BRAM et lit le premier mot de 32-bit qui sélectionne le mode de fonctionnement (avec ou sans compression) et détermine la taille du *bitstream*. Sans compression, les données de configuration sont directement chargées de la BRAM vers l'ICAP, alors qu'en mode avec compression, les données transitent via un bloc de décompression avant d'être envoyées vers l'ICAP. Les données de configuration sont chargées dans l'ICAP en utilisant des transferts *burst* sans interruption, ce qui permet d'obtenir le débit maximal de reconfiguration. Le signal "Finish" indique la fin de la reconfiguration et la bloc UReC désactive l'utilisation de la BRAM et de l'ICAP pour réduire la consommation.

4.1.2 Gestion de la reconfiguration

Au niveau système, le gestionnaire de reconfiguration s'occupe de trois tâches : le préchargement du *bitstream*, le contrôle de la reconfiguration et l'adaptation de la fréquence. Le processeur *MicroBlaze* est utilisé dans ce but mais d'autres solutions peuvent être utilisées, notamment un bloc matériel dédié pour réduire l'énergie consommée.

Le gestionnaire de reconfiguration est en charge de la lecture du fichier de bitstream dans la mémoire externe, il vérifie l'entête du fichier de bitstream partiel et charge ensuite la taille du bitstream suivi des données de configuration dans une mémoire BRAM. Le premier mot de 32 bits contient la taille du bitstream et le mode d'opération qui détermine le type de fonctionnement du contrôleur : avec ou sans compression des données. Les données de configuration pour la PRR donnée suivent ce premier mot. Le bitstream partiel contient un préambule qui définit les attributs tels que le nom du fichier, l'identification du FPGA et la taille du bitstream. Après ce préambule, le bitstream contient les données de configuration.

Le gestionnaire contrôle la reconfiguration en pilotant le signal de départ ("Start") pour lancer la procédure et reçoit le signal de fin ("Finish") lorsqu'elle est terminée. Contrairement à *xps_hwicap* où le *MicroBlaze* est occupé durant la reconfiguration, ici le *MicroBlaze* lance seulement UPaRC et est libre lors de la procédure. Ceci permet de beaucoup réduire la consommation énergétique comparé à *xps_hwicap*. De plus un petit bloc matériel peut jouer le rôle du gestionnaire de reconfiguration et réduire encore l'impact énergétique.

Le gestionnaire de reconfiguration analyse les différentes contraintes extérieures (performance, consommation en puissance et en énergie notamment) pendant le fonctionnement et choisit la fréquence appropriée pour satisfaire ces contraintes en

paramétrant le bloc DyCloGen. DyCloGen est un gestionnaire d'horloge capable de modifier dynamiquement la fréquence de l'horloge en sortie en fonction de la demande. La modification de l'horloge est contrôlée par deux facteurs : multiplieur et diviseur de fréquence. La variation de la fréquence de l'horloge permet de changer la puissance consommée et de s'adapter à différentes contraintes de fonctionnement. Le fonctionnement à fréquence élevée permet d'obtenir des performances importantes mais augmente la puissance consommée, la fréquence appropriée doit donc être sélectionnée par le gestionnaire en fonction des contraintes de performance et de consommation. De plus la fréquence maximale étant très élevée, la lecture de la BRAM ainsi que la reconfiguration ne sont pas garanties sans erreurs. Cette fréquence élevée doit être utilisée si la reconfiguration a été validée dans les mêmes conditions (circuit, température, tension d'alimentation), or ces conditions peuvent varier et nécessiter l'ajustement de la fréquence de reconfiguration pour éviter les erreurs.

4.1.3 Implémentation et performances

Le système présenté sur la Figure 4.1 a été implémenté sur deux plateformes disposant d'un FPGA Xilinx FPGA : une plateforme ML506 [87] avec un Virtex-5 XC5VSX50T [88] et une plateforme ML605 [89] avec un Virtex-6 XC6VLX240T [90] en utilisant la suite d'outils Xilinx EDK 13.2. Les ressources requises pour les blocs de UPaRC sont présentées dans le tableau 4.1.

TABLE 4.1: Ressources du FPGA requises par les blocs élémentaires de UPaRC

Bloc	Virtex-5 (slices)	Virtex-6 (slices)
DyCloGen	24	18
UReC	26	26
Décompresseur	1035	900

Les ressources requises pour les modules DyCloGen et UReC sont relativement faibles. Le décompresseur consomme une plus grande quantité de ressources à cause du traitement des données nécessaires et de la vitesse de décompression élevée permettant un débit de données important, plus de 1 GB/s. Par la suite de ces travaux de thèse, nous utiliserons UPaRC sans compression du bitstream pour limiter la consommation lors de la reconfiguration. Le décompresseur n'est plus implémenté dans UPaRC par la suite.

Avec la plateforme ML506, UPaRC atteint 362.5 MHz (avec $F_{in} = 100\ MHz$, $M = 29$ et $D = 8$ pour la configuration de DyCloGen), permettant un débit de re-

configuration de 1433 MB/s. UPaRC est testé sur plusieurs Virtex-5 XC5VSX50T FPGAs. La fréquence maximale est de 362.5 MHz pour que la reconfiguration se déroule avec succès dans nos conditions d'expérimentation (tension cœur par défaut : 1 V, température ambiante de 20°C). La Figure 4.2 situe le débit proposé par UPaRC par rapport aux contrôleurs de reconfiguration de l'état de l'art. Les tests dans les mêmes conditions sur un Virtex-6 XC6VLX240T ont montré que 362.5 MHz n'est pas une fréquence de fonctionnement permettant la reconfiguration. La reconfiguration à la fréquence de 300 MHz a été testée avec succès.

La taille de la mémoire BRAM pour le préchargement du bitstream est de 256 KB. En utilisant la compression, cette quantité de mémoire permet approximativement (en fonction du taux de compression) de stocker un bitstream de 992 KB, ce qui représente plus de 40% du bitstream complet du Virtex-5 choisi pour l'expérience (2444 KB). Ceci signifie que le mode de préchargement avec compression peut prendre en charge des PRRs qui occupent près de la moitié des ressources du FPGA. En mode de préchargement avec compression, le bloc de compression a une sortie de deux mots par cycle d'horloge (largeur de données de 64 bits), et fonctionne à une fréquence maximale de 126 MHz ce qui permet un débit de reconfiguration de 1.008 GB/s. La fréquence de fonctionnement du décompresseur est différente du bloc UReC et de la reconfiguration. Dans ce cas, UPaRC est limité par le bloc de décompression et ne peut pas fonctionner à sa fréquence maximale. La fréquence de reconfiguration maximale alors obtenue est proche de 255 MHz (la largeur des données de reconfiguration est de 32 bits).

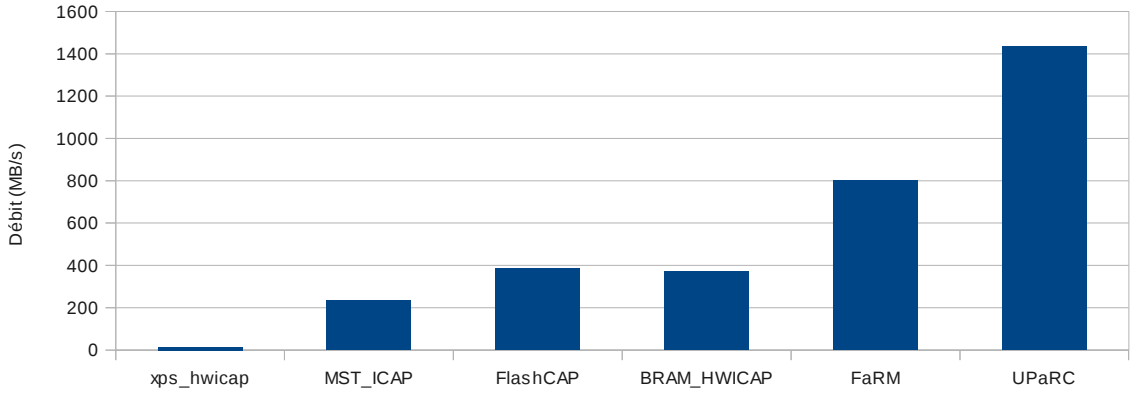

FIGURE 4.2: Comparaison du débit des contrôleurs de reconfiguration.

4.1.4 Analyse de puissance

Le débit de reconfiguration et la taille du bitstream ne sont pas les seules préoccupations majeures. La consommation en puissance (et énergie) doit être considérée, notamment dans les systèmes embarqués contraints par un budget énergétique ou

par un maximum de puissance, pour éviter les pics de puissance et assurer le bon fonctionnement du système. L'efficacité énergétique du contrôleur de reconfiguration est elle aussi importante pour permettre et favoriser l'utilisation de la reconfiguration dans les systèmes embarqués. Pour ces raisons, l'approche que nous avons suivie est expérimentale et se base sur des mesures réelles effectuées sur la plateforme Xilinx ML605 contenant un FPGA Virtex-6. La ML605 inclut les ressources nécessaires à la mesure du courant du cœur du FPGA. Il est alors possible de mesurer le consommation du Virtex-6 ce qui n'est pas possible avec la plateforme ML506. La procédure de mesure du courant est identique à celle utilisée pour les expérimentations précédentes sur la plateforme ML550 (section 2.2).

La capacité des connexions est une composante de la puissance dynamique, donc pour la réduire le contrôleur de reconfiguration doit être contraint en surface. Grâce à la faible empreinte sur la surface de UPaRC, la puissance et l'énergie consommées pendant la reconfiguration sont très faibles. Puisque *xps_hwicap* proposé par Xilinx est le seul contrôleur de reconfiguration disponible pour tous les concepteurs, nous comparons la consommation d'énergie de UPaRC avec *xps_hwicap* dans les mêmes conditions et en utilisant un *MicroBlaze* fonctionnant à 100 MHz avec un bitstream de 216.5 kB préchargé dans 256 kB de BRAM. Sans optimisation du processeur, nous sommes parvenus à un débit de reconfiguration de 1.5 MB/s pour une puissance de 45 mW soit une consommation énergétique de 30 $\mu J/kB$. Dans les mêmes conditions, avec un *MicroBlaze* en tant que gestionnaire de la reconfiguration, UPaRC (sans compression) consomme seulement 0.66 $\mu J/kB$ ce qui est 45 fois plus efficace que dans le cas où *xps_hwicap* est utilisé.

De plus, UPaRC peut fonctionner à différentes fréquences d'horloge. Cette fonctionnalité permet de respecter les contraintes de performances avec différentes caractéristiques de consommation de puissance pendant le fonctionnement. La Figure 4.3 est un tracé de la puissance consommée pendant la reconfiguration d'un bitstream non compressé de 216.5 kB pour différentes fréquences d'horloge, de 50 MHz à 300 MHz qui est la fréquence de fonctionnement maximale garantie pour les BRAMs. Sur ces courbes, le pic de puissance avant le temps zero est causé par l'activité du gestionnaire de la reconfiguration (*MicroBlaze*) et dont la puissance et la durée sont identiques pour toutes les fréquences de fonctionnement de UPaRC, puisque la fréquence du gestionnaire n'est pas modifiée. Puis la procédure de reconfiguration commence, toutes les données de configuration sont copiées depuis la BRAM vers l'ICAP. Cette activité augmente la consommation immédiatement après l'activation du signal "Start". Une fois que la reconfiguration est terminée, la puissance consommée diminue jusqu'à la puissance idle.

La consommation à 50 MHz est de 183 mW, correspondant à un temps de recon-

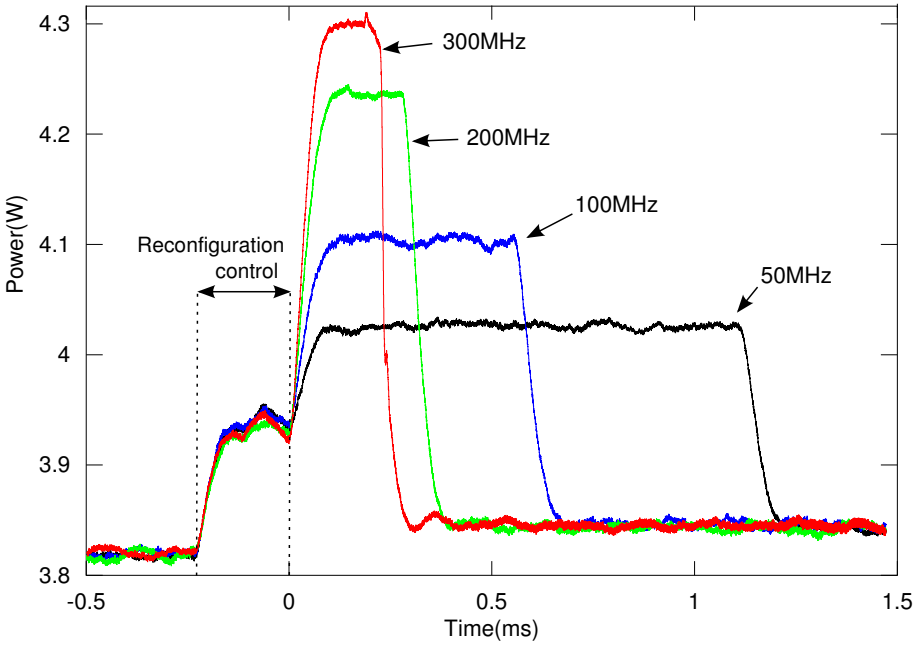

FIGURE 4.3: Consommation du cœur du FPGA pendant la reconfiguration en utilisant UPaRC avec différentes fréquences sur un Virtex-6. Seul un *MicroBlaze* et UPaRC sont implémentés.

figuration de 1.1 ms. À 100 MHz, la consommation en puissance est de 259 mW pendant 550 μs. Nous pouvons remarquer que lorsque la fréquence est doublée, le temps de reconfiguration n'est pas divisé par deux. Ceci est causé par le fait que le gestionnaire de reconfiguration est implémenté avec une attente active sur le signal "Finish". De plus celui-ci provoque une consommation de 100 mW environ pendant tout le temps de la reconfiguration. À 200 MHz, la consommation en puissance est de 394 mW pendant 270 μs. Finalement, la reconfiguration à 300 MHz consomme 453 mW pendant 180 μs.

Le gestionnaire de reconfiguration n'est actuellement pas optimisé pour réduire la consommation. Le gestionnaire attend la fin de la reconfiguration en scrutation. Ceci consomme plus de puissance que la reconfiguration gérée par UPaRC. Cette puissance est constante pour toutes les fréquences de fonctionnement de UPaRC et explique pourquoi l'énergie diminue lorsque la fréquence augmente. Avec un gestionnaire étudié pour limiter la puissance consommée pendant l'attente de la reconfiguration, l'énergie consommée pendant la reconfiguration devrait être sensiblement la même pour chaque fréquence.

La puissance augmente proportionnellement avec l'augmentation de la fréquence de fonctionnement. La solution qui permet donc de réduire la consommation et limiter les pics de puissance est la fréquence la plus faible permettant de maintenir les contraintes de temps de l'application. La fréquence est ajustée dynamiquement,

en utilisant DyCloGen, pour satisfaire à la fois les contraintes de performances et diminuer la puissance à la volée si le système le nécessite.

Nous pouvons noter que les courbes de puissance ne présentent pas de variation de la puissance durant la reconfiguration comme présenté lors de la construction du modèle de consommation dans le chapitre précédent. Nous souhaitons mesurer principalement la puissance du contrôleur de reconfiguration ainsi les données du bitstream contiennent les mêmes informations que celles déjà configurées. Il n'y a donc pas de changement de la consommation *idle* et la distance de Hamming est toujours nulle et ne provoque pas de variations de la consommation.

Cette section a présenté un contrôleur de reconfiguration, UPaRC, qui améliore significativement le débit de reconfiguration ainsi que ses performances énergétiques. De plus UPaRC permet d'adapter la fréquence de reconfiguration pour offrir un degré de liberté plus élevé, pendant le fonctionnement, pour respecter les contraintes de temps d'exécution et de puissance d'un système reconfigurable. La vitesse de reconfiguration de UPaRC rend maintenant la reconfiguration possible dans des systèmes rapides avec des contraintes de temps strictes. La faible consommation de UPaRC ouvre des possibilités de choix de reconfiguration et d'implémentation dans le but de minimiser l'énergie. Couplé avec les différentes solutions d'implémentation proposant un compromis temps-énergie-surface présenté dans le chapitre 2 et l'effacement de la configuration pour minimiser les fuites de courant, UPaRC permet d'augmenter la flexibilité des architectures reconfigurables avec l'objectif de réduire la consommation énergétique globale ou d'améliorer les performances.

4.2 Modélisation de systèmes hétérogènes reconfigurables

Cette section propose une modélisation d'une application implémentée sur une architecture multiprocesseur incluant des accélérateurs matériels avec reconfiguration dynamique partielle. La modélisation introduit l'ensemble des paramètres nécessaires pour l'exploration des solutions d'implémentation.

Dans un premier temps le modèle du système sur puce est présenté, suivi d'une formalisation des caractéristiques des tâches et de leurs implémentations.

4.2.1 Modélisation de l'architecture

Les systèmes sur puce sont principalement basés sur une architecture hétérogène incluant à la fois des ressources pour l'exécution logicielle (processeurs CPUs, processeurs graphiques GPUs ou processeurs de traitement du signal DSPs principale-

TABLE 4.2: Liste des paramètres utilisés dans la modélisation des systèmes sur puce

Variable	plage de valeurs	Définition
N^{CPU}	$\in \mathbb{N}^*$	Nombre de ressources d'exécution logicielle (processeurs)
N^{PRR}	$\in \mathbb{N}^*$	Nombre de régions reconfigurables dynamiquement (PRR)
N^{EU}	$= N^{PRR} + N^{CPU}$	Nombre total de ressources (EU—Execution Unit)
EU_j	$\forall j = 1, ..., N^{EU}$	La $j^{ième}$ ressource du système sur puce
SoC	$= \{EU_j\}$	Un système sur puce est un ensemble de ressources d'exécution
EU_j^{size}	$\in \mathbb{N}^*$	Taille de EU_j en slices (ou autre unité élémentaire)
P_j^{empty}	$\in \mathbb{R}^+$	Consommation de la ressource lorsque aucune tâche est configurée
T^{1slice}	$\in \mathbb{R}^+$	Temps requis pour reconfigurer une slice
$P_{controller}$	$\in \mathbb{R}^+$	La puissance consommée par le contrôleur de reconfiguration

ment) et des ressources matérielles pour améliorer les performances (accélérateurs matériels dédiés et régions reconfigurables qui apportent de la flexibilité).

Notre proposition de modélisation intègre ces différentes ressources. Les variables et paramètres qui sont utilisés pour spécifier le système sur puce sont présentées dans le tableau 4.2 et sont expliqués par la suite. La proposition s'appuie sur la modélisation précédemment effectuée dans la section 2.5.

4.2.1.1 Caractérisation des ressources

Deux types de ressources (notées EU pour *Execution Unit* par la suite) sont considérés dans un système sur puce : les matrices reconfigurables et les processeurs. La matrice reconfigurable est composée d'une quantité N^{PRR} de régions reconfigurables partiellement et cette composition est considérée comme statique pendant l'exécution. Le nombre de processeurs est quant à lui défini par N^{CPU}.

Les unités telles que les FPGAs actuels incluent maintenant un ou plusieurs cœur(s) de processeur. Les cœurs "logiciels" (cœur processeur implémenté dans une ressource reconfigurable) sont une autre possibilité pour qu'un système reconfigurable dispose d'une ressource processeur. Les accélérateurs matériels non reconfigurables ne sont pas explicitement modélisés car il s'agit d'un cas particulier de PRR qui est préconfigurée et jamais reconfigurée. Ils peuvent donc être représentés par une PRR sur laquelle une seule tâche est instanciable.

4.2.1.2 Caractéristiques de consommation idle

La consommation *idle* du système sur puce peut être estimée par

$$P^{SoCidle} = \sum_j P_j^{empty}. \tag{4.1}$$

où P_j^{empty} est la puissance *idle* et prend en compte la puissance statique et dynamique de chaque EU lorsqu'elle n'est pas configurée (c'est-à-dire *blank* pour une PRR) et n'exécute aucune tâche. Cette puissance est directement liée à la taille de la ressource et contient la consommation de l'arbre d'horloge, des processeurs en *idle* ou des autres circuits spécifiques pour les EU concernées.

4.2.1.3 Caractéristiques de la reconfiguration dynamique

Le temps de reconfiguration dépend du débit de reconfiguration et de la taille du bitstream, il est défini par la variable T^{1slice}. La consommation de la reconfiguration est estimée en utilisant le modèle à gros grain présenté dans la section 3.2.1. La puissance consommée par la reconfiguration est dépendante du contrôleur de reconfiguration et de la technologie du FPGA, elle est déterminée par la variable $P_{controller}$.

Comme présenté dans le chapitre 3, les régions reconfigurables sont divisées en blocs, les *frames*. Cependant cette quantité n'est pas toujours facile à obtenir et on parle plus communément des *slices* contenant les CLBs (terme emprunté à Xilinx). Les quantités utilisées sont en général faciles à obtenir avec les outils de conception. Une *slice* est alors considérée, à ce niveau, comme le grain élémentaire de reconfiguration pour un système reconfigurable. Une PRR est constituée d'un nombre entier de *slices* et la taille de la PRR_j est définie par EU_j^{size}. Le lien avec la taille du bitstream est direct, le bitstream contient un nombre défini d'octets par *slice*.

L'utilisation de la reconfiguration dynamique pour une application complète nécessite la mise en place d'un gestionnaire de reconfiguration (service d'un système d'exploitation par exemple) capable d'évaluer l'ordonnancement des tâches pour assurer le bon fonctionnement de l'application. Ce choix d'ordonnancement et d'implémentation des tâches peut être effectué dans le but de satisfaire des contraintes de temps ou de réduction de l'énergie. La reconfiguration permet d'envisager la préemption et la relocalisation des tâches sur les ressources reconfigurables, un système préemptif offre un plus grand degré de flexibilité lors de l'exécution d'une application. Néanmoins, la préemption des tâches nécessite une sauvegarde de contexte et cette opération est coûteuse notamment pour les systèmes reconfigurables. Ces délais liés à la préemption augmentent considérablement la difficulté de prédire l'ordonnancement des tâches et la complexité pour l'exploration. La sauvegarde et la restauration de contexte sont aujourd'hui des opérations délicates et complexes (voire impossibles selon l'architecture) à mettre en œuvre dans les systèmes reconfigurables. Nous avons alors choisi de ne pas prendre en compte cette option dans notre modélisation et stratégie d'exploration.

TABLE 4.3: Liste des paramètres utilisés pour la modélisation des applications

Variable	Plage de valeurs	Définition
N^T	$\in \mathbb{N}^*$	Nombre de tâches d'une application
T_i	$\forall i = 1, ..., N^T$	La $i^{ième}$ tâche d'une application
$\mathcal{G}$	$= \{T_i\}$	Le graphe de dépendance des tâches de l'application
N_i^{imp}	$\in \mathbb{N}^*$	Nombre d'implémentations disponibles pour T_i
$T_{k,i}$	$\forall k = 1, ..., N_i^{imp}$	La $k^{ième}$ implémentation de T_i
$C_{k,i}$	$\in \mathbb{R}^+$	Le temps d'exécution de $T_{k,i}$
$E_{k,i}$	$\in \mathbb{R}^+$	L'énergie consommée par $T_{k,i}$
$P_{k,i}^{idle}$	$\in \mathbb{R}^+$	Puissance idle consommée par $T_{k,i}$
$I_{k,i,j}$	$\in \{0, 1\}$	Définit si $T_{k,i}$ est instanciable sur EU_j

4.2.2 Modélisation de l'application

Le tableau 4.3 résume tous les paramètres utilisés pour la modélisation de l'application. La description concerne en premier lieu le graphe de tâches puis la définition de l'instanciation des tâches sur les ressources. Enfin les caractéristiques des tâches, temps d'exécution et énergie principalement, sont précisées.

4.2.2.1 Graphe de dépendance des tâches

Une application peut être définie par un ensemble de tâches et ces tâches ont des dépendances entre-elles. Ces dépendances, généralement de données, doivent être décrites pour effectuer avec succès les traitements et obtenir le résultat de l'application escompté. Le graphe de tâches, $\mathcal{G}$, est composé d'un ensemble de N_T tâches et spécifie les dépendances des tâches, représentées sous la forme d'un graphe acyclique orienté et sa matrice d'adjacence. Cette représentation permet d'exposer le parallélisme entre les tâches et de l'exploiter facilement.

4.2.2.2 Instanciation des tâches

Avec le support des ressources reconfigurables, une tâche peut avoir différentes implémentations exploitant différentes ressources. Cette particularité met à disposition plusieurs versions d'une tâche. Le niveau de parallélisme est un bon moyen de générer différentes implémentations d'une tâche et conduit à obtenir différents compromis temps/énergie/surface. L'utilisation du déroulage de boucles, présenté dans le chapitre 2 par exemple, est une technique qui peut être utilisée pour générer ces différentes implémentations. Grâce à la synthèse de haut niveau, le coût de développement de plusieurs implémentation est faible.

Une fois que le déploiement d'une tâche est défini, chaque implémentation d'une tâche T_i, notée $T_{k,i}$ doit être caractérisée. La variable $I_{k,i,j}$ est définie en fonction des

possibilités d'implémentation de $T_{k,i}$ sur la ressource EU_j. La valeur de $I_{k,i,j}$ est 1 si la tâche $T_{k,i}$ est instanciable sur la ressource EU_j, ou 0 sinon.

4.2.2.3 Temps d'exécution, puissance et énergie

Le temps d'exécution de chaque tâche T_i dépend de son implémentation, $T_{k,i}$, et il est défini par la variable $C_{k,i}$. La consommation énergétique de chaque tâche T_i est elle aussi dépendante de son implémentation et elle est définie par la variable $E_{k,i}$ pour la $k^{ième}$ implémentation de la tâche T_i. Chaque tâche matérielle lorsqu'elle est configurée a une consommation en puissance même si elle n'est pas utilisée. Il s'agit de la consommation *idle*, notée $P^{idle}_{k,i}$ pour la $k^{ième}$ implémentation de la tâche T_i. Ainsi la puissance *idle* globale consommée par PRR_j lorsque $T_{k,i}$ est configurée est $P^{idle}_{k,i} + P^{empty}_j$. $P^{idle}_{k,i}$ prend en compte la puissance consommée liée directement à l'implémentation de la tâche. Les autres puissances statique et *idle* présentes lorsque l'EU n'est pas configurée sont liées à l'EU en question et définies précédemment par P^{empty}_j.

Bien sûr toutes les variables de puissance et d'énergie définies précédemment dépendent des paramètres classiques tels que la tension d'alimentation du cœur, la fréquence de fonctionnement et la température notamment. Ces paramètres sont considérés comme étant constants pour simplifier la notation et les modèles. Néanmoins, les techniques DVFS qui sont utilisées couramment dans les processeurs actuels, peuvent être spécifiées comme étant des implémentations différentes d'une tâche dans la même EU avec une énergie et temps d'exécution différents.

4.3 Algorithme d'exploration

À partir de la caractérisation de l'application et du système, nous proposons un algorithme, nommé PREexplorer, pour l'exploration exhaustive des choix d'implémentation et d'ordre d'exécution des tâches d'une application complète. Parmi l'ensemble des résultats, les meilleurs choix d'ordonnancement et d'allocation des tâches permettant de réduire la consommation d'énergie ou le temps d'exécution par exemple sont extraits. La Figure 4.4 est une représentation visuelle de l'algorithme. Les sections suivantes explicitent les différentes étapes de cette figure.

4.3.1 Paramètres d'entrée

Les informations sur l'application, les tâches et le graphe de tâches ainsi que les ressources disponibles (Figure 4.4–1, pour l'étape 1 de la Figure 4.4) sont nécessaires au fonctionnement de l'algorithme. Ces informations sont statiques et définies avant l'exploration, ce qui signifie que les applications n'ont pas un graphe de tâche

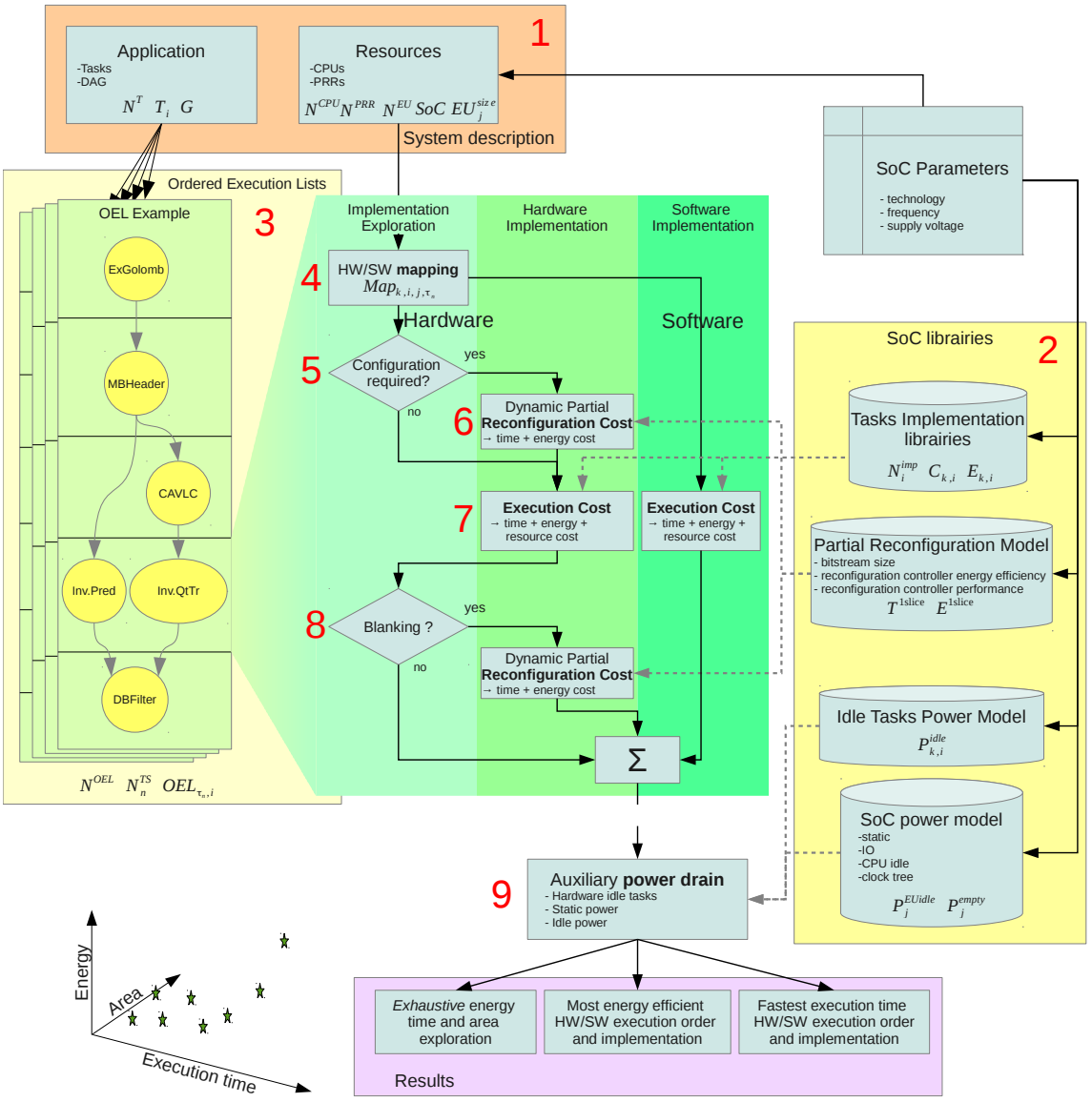

FIGURE 4.4: Représentation de l'algorithme d'exploration des choix d'implémentation et d'ordre d'exécution pour extraire la meilleure solution au niveau de l'énergie ou des performances par exemple, pour un système utilisant la reconfiguration partielle dynamique dans les systèmes hétérogènes.

TABLE 4.4: Liste de variables utilisées lors de l'exploration

Variable	Plage de valeurs	Définition
EU_{j,τ_n}^{free}	$\in \{0,1\}$	Disponibilité de la ressource EU_j
N^{OEL}	$\in \mathbb{N}^*$	Nombre de OELs extraites de $\mathcal{G}$
N_n^{TS}	$\forall n = 1, ..., N^{OEL}$	Nombre de périodes dans la $n^{ième}$ OEL
τ_n	$= 1, ..., N_n^{TS}$	période courante de la $n^{ième}$ OEL
$OEL_{\tau_n,i}$	$\in \{0,1\}$	Présence de T_i pour le période τ_n
Map_{k,i,j,τ_n}	$\in \{0,1\}$	$T_{k,i}$ implémenté sur EU_j au période τ_n

dynamique et que la répartition des ressources (notamment la définition des PRRs) doit être effectuée lors de la modélisation. L'exploration pour la caractérisation des PRRs n'est pas abordée ici et peut être effectuée à l'aide d'une autre approche, comme celle présentée dans [91].

Les modèles de puissance, d'énergie et de temps sont nécessaires pour l'exploration (Figure 4.4–2). Premièrement, la consommation d'énergie et le temps d'exécution des tâches doivent être spécifiés pour chaque implémentation sur chaque ressource existante. Ensuite un modèle de reconfiguration dynamique est requis pour estimer et prendre en compte le coût de reconfiguration. Ce modèle permet d'estimer l'énergie et le temps nécessaires pour reconfigurer une PRR en fonction de sa taille. Ces caractéristiques dépendent du contrôleur de reconfiguration utilisé et dépendent aussi de la famille de composant. Par ailleurs, lorsque des tâches configurées ne sont pas en cours d'utilisation, elles consomment quand même une puissance supplémentaire que nous avons précédemment défini comme puissance *idle*. Cette puissance dépend principalement de la surface occupée par la tâche et de la taille de la PRR concernée. Reconfigurer une PRR avec une tâche vide (blank) est une technique utilisée pour réduire la puissance *idle* et cette possibilité est analysée lors de l'exploration. Finalement d'autres paramètres liés au système sur puce sont considérés pour estimer d'autres contributions telles que l'arbre d'horloge, la puissance *idle* des processeurs ainsi que les puissances d'autres ressources spécifiques non considérées précédemment.

4.3.2 Listes d'exécution ordonnée

Pour calculer les estimations de performance et d'énergie, les différentes possibilités au niveau de l'ordre d'exécution des tâches doivent être connues. C'est le rôle de la liste d'exécution ordonnée. La liste d'exécution ordonnée (notées OEL pour *Ordered Execution List*) est un graphe de tâches renseigné avec les dates de début de tâches. L'intervalle de temps utilisé pour la modélisation est défini par les événements de début et de fin de tâche, cet intervalle est appelé "période". Chaque changement de

période τ_n correspond alors à la fin d'une tâche et au moins au début d'une autre. L'OEL est une représentation d'un ordonnancement très simple. Naturellement, si l'application permet le parallélisme entre les tâches, plusieurs OELs sont possibles. Un exemple d'OEL est donné à la Figure 4.4–3 où la tâche Intra est exécutée après la fin de l'exécution de CAVLC et en même temps que QtTr. Le tableau 4.4 récapitule les variables utilisées pour la formalisation des OELs.

4.3.3 Exploration de l'allocation

L'exploration des implémentations possibles sur les ressources (allocation) est le cœur de l'algorithme. L'algorithme analyse récursivement les coûts d'implémentation (nécessité de reconfigurer ou non) et d'exécution pour chaque période. Cette analyse est basée sur les étapes suivantes.

4.3.3.1 Choix de l'instance

L'ensemble des allocations possibles pour la période τ_n (Figure 4.4–4) est formalisé par l'expression

$$Map_{k,i,j,\tau_n} = I_{k,i,j} \times EU^{free}_{j,\tau_n} \times OEL_{\tau_n,i} \tag{4.2}$$

où EU^{free}_{j,τ_n} indique la disponibilité de la ressource EU_j. Cette variable est mise à jour à chaque fois qu'une tâche est lancée ou terminée et vaut 1 lorsque la ressource est disponible ou 0 lorsqu'une tâche s'exécute. L'objectif de cette étape de l'exploration est de trouver quelle instance k de la tâche T_i sera exécutée et sur quelle ressource EU_j. Pour pouvoir s'exécuter, $T_{k,i}$ doit être implémentable sur EU_j ($I_{k,i,j} = 1$) et cette ressource doit être disponible pour la période en cours d'exploration ($EU^{free}_{j,\tau_n} = 1$). L'algorithme explore successivement l'ensemble des possibilités d'allocations proposées par Map_{k,i,j,τ_n}.

4.3.3.2 Coût de la reconfiguration partielle

Si l'EU sélectionnée pour l'implémentation de la tâche est une ressource matérielle (PRR), la tâche implémentée est un accélérateur matériel. Une reconfiguration partielle de la ressource est requise sauf si la tâche sélectionnée est déjà configurée sur la PRR en question (Figure 4.4–5). Lorsque la reconfiguration est nécessaire, la consommation en énergie et le temps requis sont calculés (Figure 4.4–6) selon les

équations suivantes :

$$T^{conf} = \sum_{k,i,j,\tau_n} T^{1slice} \times EU_j^{size} \times Map_{k,i,j,\tau_n} \quad (4.3)$$

$$E^{conf} = \sum_{k,i,j,\tau_n} P_{controller} \times T^{1slice} \times EU_j^{size} \times Map_{k,i,j,\tau_n} \quad (4.4)$$
$$\forall\, j = 1, \ldots, N_{PRR}.$$

Il est à noter que contrairement à l'équation 3.2, la puissance P_{before}^{idle} n'intervient pas ici car la puissance *idle* est calculée globalement dans la suite de l'exploration.

4.3.3.3 Coût associé à l'exécution des tâches

Finalement, la tâche débute son exécution ce qui engendre une consommation pendant tout le temps de cette exécution. Le temps d'exécution et la consommation sont estimés (Figure 4.4–7) par $C_{k,i}$ et $E_{k,i}$. Le temps d'exécution total est défini par la somme des maximums de temps d'exécution pour chaque période tandis que la consommation en énergie est la somme de l'énergie de chaque tâche :

$$T^{exec} = \sum_{k,i,j,\tau_n} \max_{\tau_n}(C_{k,i} \times Map_{k,i,j,\tau_n}) \quad (4.5)$$

$$E^{exec} = \sum_{k,i,j,\tau_n} E_{k,i} \times Map_{k,i,j,\tau_n}. \quad (4.6)$$

4.3.3.4 Effacement

Si la PRR n'est pas utilisée pendant un certain temps, l'effacement de sa configuration (*blank*) est probablement un choix intéressant du point de vue de la consommation d'énergie (Figure 4.4–8). Toutefois, si l'effacement est effectué, il ajoute le coût de reconfiguration de la ressource. Il est donc important d'évaluer si cet effacement amène une réduction de la consommation. L'évaluation de l'intérêt de l'effacement est décrite dans le chapitre 2 par l'équation 2.16. Cependant, si la décision d'effacer peut être prise ici, les deux possibilités (effacement ou non) sont explorées car elles peuvent satisfaire différents objectifs, notamment celui de réduction du temps d'exécution. Par exemple si une tâche est exécutée deux fois au cours d'une application, le fait de ne pas effacer la configuration de la ressource permet d'éviter une autre reconfiguration. De même, si le contrôleur de reconfiguration (qui est généralement unique pour l'ensemble des ressources matérielles) est fortement sollicité sur l'ensemble du système, demander un effacement d'une ressource entraîne des délais supplémentaires.

Cette exploration est effectuée pour chaque tâche présente dans la période courante ce qui permet de prendre en charge les tâches pouvant s'exécuter en parallèle

sur des ressources différentes. Si aucune unité d'exécution n'est disponible pour implémenter une tâche, ceci signifie que l'ordonnancement proposé par la liste d'exécution n'est pas possible dans les conditions choisies pour cette solution. L'exploration de cette solution est stoppée et on procède à l'exploration d'un autre choix d'allocation.

L'exploration des allocations est effectuée pour chaque période de la liste d'exécution ordonnée ainsi que pour chaque liste d'exécution ordonnée. Ainsi, d'autres solutions ayant un ordonnancement et des allocations différents sont explorées.

4.3.4 Puissances auxiliaires

À l'étape 9 de la Figure 4.4, les consommations de puissance auxiliaires sont calculées. La puissance statique du système sur puce, la puissance *idle* incluant l'arbre d'horloge, les puissances *idle* des processeurs ou des accélérateurs statiques provoquent une consommation supplémentaire qui est estimée et ajoutée. Il est nécessaire de prendre en compte la consommation des autres blocs et périphériques (le cas échéant : parties analogiques, générateur d'horloge, entrées et sorties du SoC...) comprise dans $P^{SoCidle}$ pour estimer la puissance totale du système sur puce.

$$E^{drain} = P^{SoCidle} \times (T^{exec} + T^{conf}) + \sum_{k,i,j,\tau_n} P^{idle}_{k,i} \times Map_{k,i,j,\tau_n} \times T^{idle}_{i}. \tag{4.7}$$

4.3.5 Résultats

Tous les temps d'exécution et coûts énergétiques sont sommés pour chaque allocation et ordonnancement associé.

$$E^{tot} = E^{conf} + E^{exec} + E^{drain} \tag{4.8}$$

$$T^{tot} = T^{conf} + T^{exec} \tag{4.9}$$

$$\forall\, HW/SWMapping,\ \forall\, OEL$$

Puis tous les temps requis, énergie consommée, profil de puissance, occupation des ressources sont sauvegardés pour chaque solution d'implémentation trouvée. Les résultats, exhaustifs sont affichés pour mettre en évidence les besoins en temps, énergie et surface de chaque solution. À partir de ces résultats, quelques solutions sont extraites dont la solution la plus efficace en énergie et la solution la plus performante (temps d'exécution le plus faible). Le profil de puissance et l'occupation des ressources peuvent être affichés avec l'ordre d'exécution et les implémentations correspondantes. Parmi ces résultats, deux autres solutions sont également mises

en avant : la première est une solution avec une exécution entièrement logicielle (notée *Full SW*), sans accélérateur matériel. La seconde est une solution basée entièrement sur des accélérateurs matériels statiques (notée *Static HW*), c'est-à-dire sans utiliser de reconfiguration dynamique lorsque les tâches sont matérielles, sinon l'exécution est logicielle. Ces résultats permettent de comparer les performances de ces solutions avec les solutions exploitant la reconfiguration dynamique partielle des ressources matérielles.

L'exploration est exhaustive au niveau des possibilités d'allocation des tâches afin d'obtenir une solution optimale en fonction des contraintes décrites. Pour cette raison, la complexité de l'algorithme d'exploration est exponentielle et dépend du nombre de tâches et du nombre de ressources. L'ordonnancement par OEL est une approche simple dont l'utilisation permet de limiter la complexité de l'exploration. La contrepartie est qu'il est certainement possible de trouver dans certains cas un ordonnancement qui améliore les résultats obtenus lors de l'exploration.

4.4 Étude de cas : le décodeur H.264

Afin de valider la modélisation et l'exploration, nous prenons une application de traitement vidéo en cas d'étude. Cette application est un décodeur vidéo du standard MPEG-4 AVC/H.264, norme couramment utilisée car elle permet un codage efficace pour fournir une vidéo de haute qualité à un débit très réduit. Le code utilisé est une version modifiée du code de référence de l'UIT - ITU [92] pour une implémentation sur FPGA. L'application est un bon cas d'étude car les quantités de traitements sont importantes et sont amenées à être utilisées dans des systèmes embarqués à énergie et temps contraints.

4.4.1 Description du système

L'application de décodage vidéo est composée de six tâches appelées codage exponentiel-Golomb (notée ExGolomb), MBHeader, codage inverse CAVLC (*Context-adaptive Huffman variable-length coding*, notée Inv.CAVLC), transformée inverse (notée Inv.QTr.), prédiction inverse temporelle et spatiale (prédiction *intra* et *inter*, notée Inv.Pred.) et le filtrage anti-blocs (noté DBFilter). Un *profiling* du code original met en évidence les tâches à accélérer, il s'agit de DBFilter, Inv.CAVLC et Inv.QTr qui représentent 76% du temps d'exécution du processeur. Afin d'augmenter les performances du décodage vidéo, ces fonctions sont accélérées matériellement. Les blocs matériels sont générés à partir de la méthode présentée dans le chapitre 2, à l'aide de synthèse de haut niveau et de transformations de code de type déroulage de boucles pour obtenir plusieurs versions avec des caractéristiques différentes. Les blocs peuvent être

TABLE 4.5: Paramètres des tâches H.264 (tâches logicielles)

	SW_{ex}	
Tâche	$T_{ex}(ms)$	$E_{ex}(mJ)$
ExGolomb	5	2.23
MBHeader	4.92	2.19
Inv.CAVLC	22.06	9.8
Inv.QTr.	10.19	4.5
Inv.Pred.	10.77	4.8
DBFilter	34.98	15.6

connectés sur le bus système (AXI) pour permettre le transfert des données ainsi que le contrôle de la tâche (démarrage, fin). Cette méthode d'accélération du décodeur vidéo est issue de [82].

La Figure 4.5 présente le graphe de tâches de l'application qui peut être formulé par :

$$\begin{aligned} \mathcal{G} \quad = \quad & \{ExGolomb,\ MBHeader,\ Inv.CAVLC, \\ & Inv.QtTr.,\ Inv.Pred,\ DBFilter\}. \end{aligned} \tag{4.10}$$

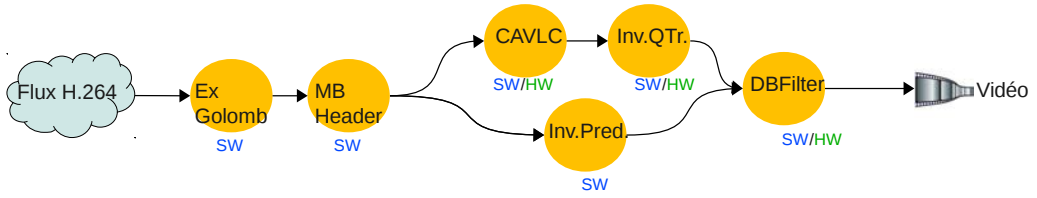

FIGURE 4.5: Graphe de tâches du décodeur H.264

Nous avons sélectionné deux solutions pour chaque tâche matérielle correspondant à la version la plus lente et à la plus rapide excepté pour Inv.CAVLC où le parallélisme est difficile à exploiter. Chaque tâche dispose d'une description logicielle et peut également être exécutée sur un processeur. Les caractéristiques des implémentations matérielles et logicielles des tâches sont présentées dans les tableaux 4.5 et 4.6. Ces paramètres sont issus d'une exécution matérielle sur un FPGA Virtex-6 LX240T (carte de développement Xilinx ML605) tandis que les paramètres de l'exécution logicielle sont obtenus à partir de l'exécution sur un processeur ARM CortexA8 à 600 MHz (BeagleBoard de Texas Instruments). Le temps et l'énergie varient en fonction des données traitées et les valeurs présentées ici sont des moyennes.

Pour cette application, nous considérons que l'exécution a lieu sur un SoC supposé composé d'un cœur ARM CortexA8 et d'une ressource matérielle supportant la reconfiguration dynamique partielle dans un composant Virtex-6. Nous avons

TABLE 4.6: Paramètres des tâches matérielles H.264 (version séquentielle et parallèle)

	HW_{seq}				HW_{par}			
Tâche	$T_{ex}(ms)$	$E_{ex}(mJ)$	$P_{idle}(mW)$	Slices	$T_{ex}(ms)$	$E_{ex}(mJ)$	$P_{idle}(mW)$	Slices
Inv.CAVLC	14.05	0.25	55.1	3118	–	–	–	–
Inv.QTr.	4.92	0.06	34.2	1056	3.93	0.05	42.2	1385
DBFilter	3.14	0.02	33.4	686	3.11	0.02	40.3	1869

TABLE 4.7: Liste des variables pour l'exploration de l'application H.264 dans le cas étudié

Variable	Valeur	Variable	Valeur
N^{CPU}	1	N^{PRR}	2
N^{EU}	1+2=3	EU^{size}_{CPU}	-
EU^{size}_{PRR1}	$1200 slices$	EU^{size}_{PRR2}	$3200 slices$
P^{empty}_{CPU}	100 mW	P^{empty}_{PRR1}	50 mW
P^{empty}_{PRR2}	133 mW	T^{1slice}	41 μs
$P_{controller}$	20 mW	N^{T}	6

choisi d'utiliser dans un premier temps un contrôleur de reconfiguration plus rapide (40 MB/s) que le contrôleur de Xilinx, *xps_hwicap* (15 MB/s). Ceci permet de minimiser le coût temporel de la reconfiguration. La ressource reconfigurable est découpée en deux PRRs de différentes tailles. Ce découpage est effectué de manière à ce que les tâches puissent y être configurées. Les tailles des deux PRRs, $PRR1$ et $PRR2$ sont respectivement de 1200 et 3200 slices. Toutes les tâches peuvent être configurées sur la PRR la plus grande ($PRR2$) alors que la petite PRR ($PRR1$) ne peut recevoir que les implémentations à faible parallélisme (HW_{seq}) de Inv.QTr. et DBFilter.

L'exploration est utilisée dans un premier temps pour comparer deux modèles de consommation de la reconfiguration dynamique partielle, présentés dans le chapitre précédent. La comparaison est effectuée entre le modèle à gros grain et le modèle à grain fin pour la solution d'implémentation offrant la consommation d'énergie la plus faible. Nous considérons qu'aucune tâche matérielle n'est configurée au début de l'exécution.

4.4.2 Comparaison des modèles de consommation de la reconfiguration

Les résultats d'exploration utilisant le modèle à gros grain, sont présentés Figure 4.6. Le haut de la figure représente l'implémentation des tâches matérielles et logicielles sur les ressources respectives en fonction du temps. Pour cette solution retenue,

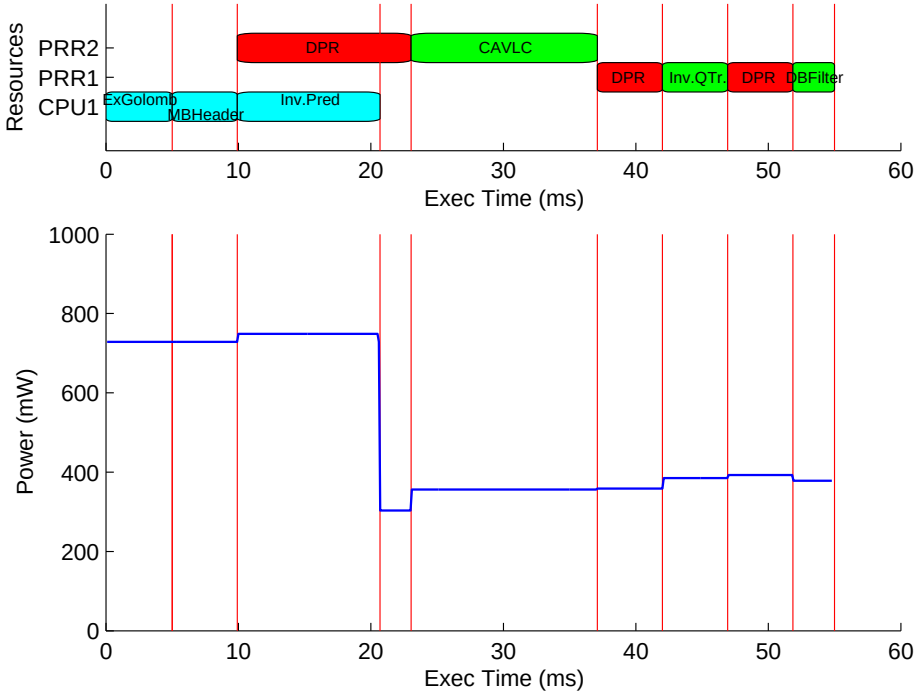

FIGURE 4.6: Implémentation et ordonnancement des tâches et profil de puissance, estimés à partir du modèle à gros grain.

trois tâches sont exécutées sur le processeur (en bleu) et les trois autres tâches sont exécutées en utilisant les deux PRRs disponibles (en vert). Une phase préliminaire de reconfiguration est nécessaire (en rouge) avant l'exécution des tâches matérielles. Les temps de reconfiguration sont importants, 13 ms pour la $PRR2$ et 5 ms pour la $PRR1$, étant donné la vitesse du contrôleur de reconfiguration utilisé et la taille des PRRs. La reconfiguration est effectuée dès que les dépendances de tâches sont satisfaites. Pour éviter de retarder l'exécution des tâches, la reconfiguration pourrait être prédite et débuter plus tôt. Cette possibilité n'est pas explorée actuellement. Malgré ces surcoûts de reconfiguration, la solution d'implémentation proposée est 37% plus rapide que la solution logicielle utilisant un seul processeur. Le bas de la Figure 4.6 représente l'estimation de la puissance consommée en fonction du temps d'exécution. Sur ce profil de puissance, la consommation du processeur est bien visible lorsque la tâche Inv.Pred. se termine (à 21 ms) et qu'il passe de l'état en cours d'exécution à l'état *idle*. La puissance consommée par les reconfigurations est elle aussi bien visible. Cette puissance, estimée à partir du modèle à gros grain, est constante durant la reconfiguration et présente avant chaque exécution matérielle à $10-23$ ms, $37-42$ ms et $47-52$ ms.

Les résultats d'exploration utilisant le modèle à grain fin sont présentés Figure 4.7. Ces résultats sont toujours extraits de la solution d'implémentation offrant la consommation d'énergie la plus faible. Le résultat d'ordonnancement et d'implémentation

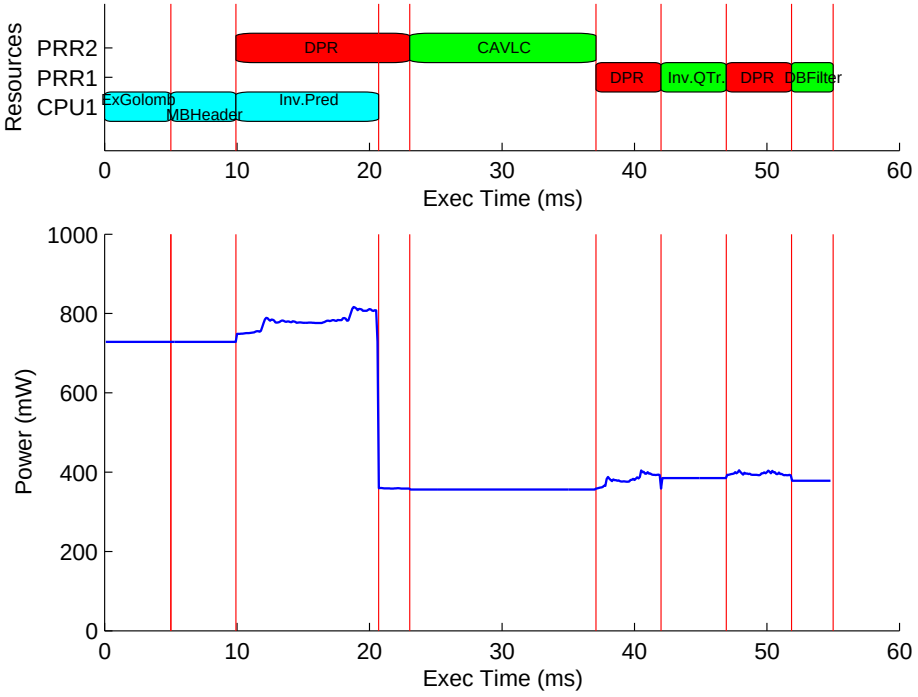

FIGURE 4.7: Implémentation et ordonnancement des tâches et profil de puissance, estimés à partir du modèle à grain fin.

TABLE 4.8: Résultats d'exploration

	Énergie (mJ)	Temps (ms)	Puissance en pic (mW)
Gros Grain	31.91	54.99	748.48
Grain Fin	32.53 (1.9%)	54.99	816.5 (+9.1%)
Exécution logicielle	39.12	87.92	445
Exécution matérielle (sans DPR)	25.1	31.11	953

des tâches est identique au modèle à gros grain de la consommation de la reconfiguration dynamique. Les deux profils de puissance des Figures 4.6 et 4.7 sont très similaires excepté certains détails résultant du niveau de détail du modèle à grain fin. Ceci met en valeur les différences entre les deux modèles ; les surconsommations estimées par la distance de Hamming sont visibles sur la puissance globale du SoC. Les sauts de consommation sont également présents, à 12 ms et 19 ms lors de la configuration de CAVLC et à 38 ms et 40 ms lors de la configuration de Inv.QTr. Les sauts de puissance ne sont pas visibles pour DBFilter puisque les puissances *idle* de Inv.QTr. et DBFilter sont très similaires (33.4 mW et 34.2 mW).

Les résultats chiffrés de ces explorations sont reportés dans le tableau 4.8 pour comparaison entre quatre solutions. Les résultats d'exploration pour deux modèles d'estimation de la consommation de la reconfiguration dynamique et les deux solutions classiques, *Full SW* et *Static HW* sont présentés.

Premièrement, la différence de consommation énergétique estimée avec les deux modèles est seulement de 0.62 mJ (2%). Cette différence est assez faible et correspond à la différence d'estimation de la puissance lors de la reconfiguration. Deuxièmement, le temps estimé de l'exécution de l'application est le même pour les deux modèles, ce qui montre que les différences énergétiques n'ont pas affecté les choix d'implémentation retenus lors de l'exploration du cas étudié. Finalement, le pic maximum de puissance estimée par le modèle à grain fin est de 68 mW plus important (9%) que le pic estimé via le modèle à gros grain. Comme mentionné dans le chapitre précédent, le modèle à grain fin est utile pour considérer des caractéristiques fines comme le pic de puissance dans ce cas. La puissance maximum consommée par le SoC est visiblement impactée par la reconfiguration dynamique, ce qui montre que celle-ci doit être prise en compte dans le développement de systèmes sensibles aux profils de puissance. C'est le cas des systèmes sur batterie pour préserver l'autonomie et éviter les pics de puissance ou des systèmes sensibles aux variations de température en fonction de la charge de travail. En revanche, le modèle à gros grain donne une estimation simple mais suffisamment précise de l'énergie (2%) pour une exploration au niveau système.

Dans ce cas d'étude, l'utilisation d'accélérateurs reconfigurables dynamiquement combinée à l'utilisation d'un processeur apporte une accélération remarquable ainsi qu'une réduction de l'énergie consommée comparé à une exécution complètement logicielle. Le taux de décodage est de 18 fps tandis que l'exécution monoprocesseur atteint 11 fps, ce qui est 64% plus rapide. Le gain énergétique est de 17% avec la reconfiguration dynamique. Cependant, l'exécution avec des accélérateurs matériels statiques (sans reconfiguration) a une consommation énergétique plus faible que la solution consommant le moins d'énergie en utilisant la reconfiguration dynamique (25.10 mJ contre 32.53 mJ). Les temps d'exécution sont respectivement de 31 ms (32 fps) pour la solution statique et de 55 ms avec reconfiguration. Il s'agit donc de 24 ms du temps d'exécution (43%) qui est utilisé pour les phases de reconfiguration. Néanmoins, il faut noter que ces résultats sont très dépendants des performances du contrôleur de reconfiguration. En utilisant un contrôleur plus rapide, la tendance s'inverse et la reconfiguration dynamique surpasse la solutions statique en consommation énergétique. Pour cette raison, l'exploration est réitérée en utilisant un contrôleur de reconfiguration plus performant, tel que UReC/UPaRC.

4.4.3 Résultats avec un contrôleur de reconfiguration optimisé

Nous cherchons à déterminer dans quelle mesure la reconfiguration dynamique permet de concurrencer les accélérateurs matériels statiques pour l'application de décodage vidéo H.264, grâce à un contrôleur permettant une reconfiguration plus rapide.

Les solutions obtenues précédemment ont des performances temporelles inférieures à une implémentation *Static HW* et une consommation d'énergie supérieure. L'exploration précédente a montré les différences entre les deux modèles de consommation de la reconfiguration dynamique partielle. La différence entre l'énergie estimée par le modèle à gros grain et le modèle à grain fin est faible (2%). Dans les explorations que nous souhaitons effectuer maintenant, l'estimation des pics de puissance n'est pas nécessaire, seuls l'énergie consommée et le temps d'exécution que nous souhaitons minimiser sont nécessaires. Le modèle à gros grain est suffisant pour obtenir un ordonnancement et une allocation pour cet objectif de minimisation.

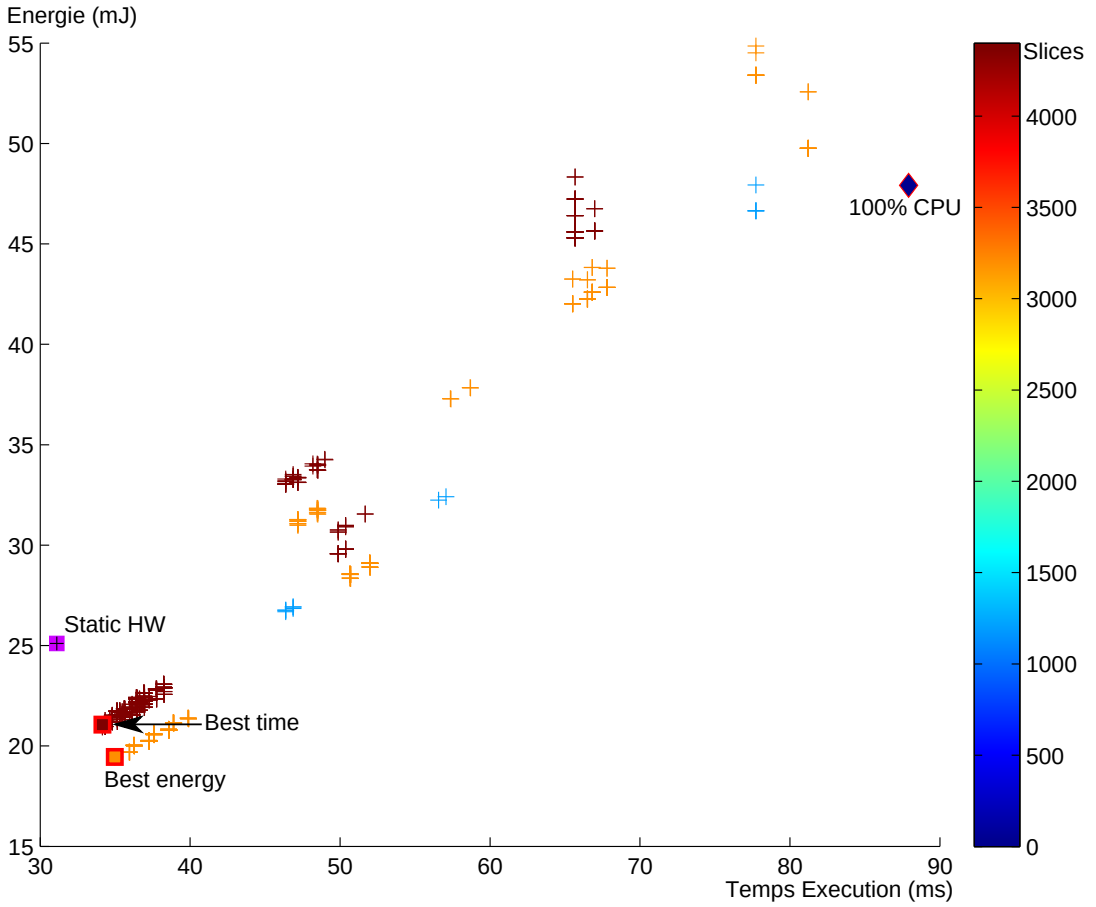

FIGURE 4.8: Ensemble des résultats obtenus en fonction des choix d'implémentation. Ces résultats sont présentés par leur énergie consommée en fonction du temps d'exécution et la surface matérielle requise est indiquée par une échelle de couleurs. Les résultats caractéristiques : exécution sur processeur uniquement (*100% CPU*), meilleur temps avec reconfiguration (*Best time*), plus faible énergie consommée avec reconfiguration (*Best energy*) et une exécution avec des accélérateurs matériels statiques (*Static HW*) sont présentés.

La vitesse de reconfiguration est plus rapide (400 MB/s) et la puissance du contrô-

leur de reconfiguration lors de la reconfiguration est $P_{controller} = 150\ mW$. La Figure 4.8 présente les résultats d'exploration. Cette figure représente les résultats de chaque solution d'ordonnancement et d'allocation explorée. On note la présence de la solution *Full SW* représentée par un losange rouge labellisé "100% CPU". Il s'agit de la solution au temps d'exécution le plus lent (88 ms) et sa consommation d'énergie est élevée (48 mJ). Les solutions utilisant une PRR sont plus rapides et ont généralement une consommation plus faible. Les solutions les plus performantes utilisent la plus grande PRR (3200 slices), afin de pouvoir implémenter toutes les tâches ayant une description matérielle, ou bien les deux PRRs (soit un total de 4400 slices). L'extraction des solutions minimisant l'énergie ou le temps d'exécution donne deux résultats très proches. Ces deux solutions sont détaillées respectivement sur les figures 4.9 et 4.10.

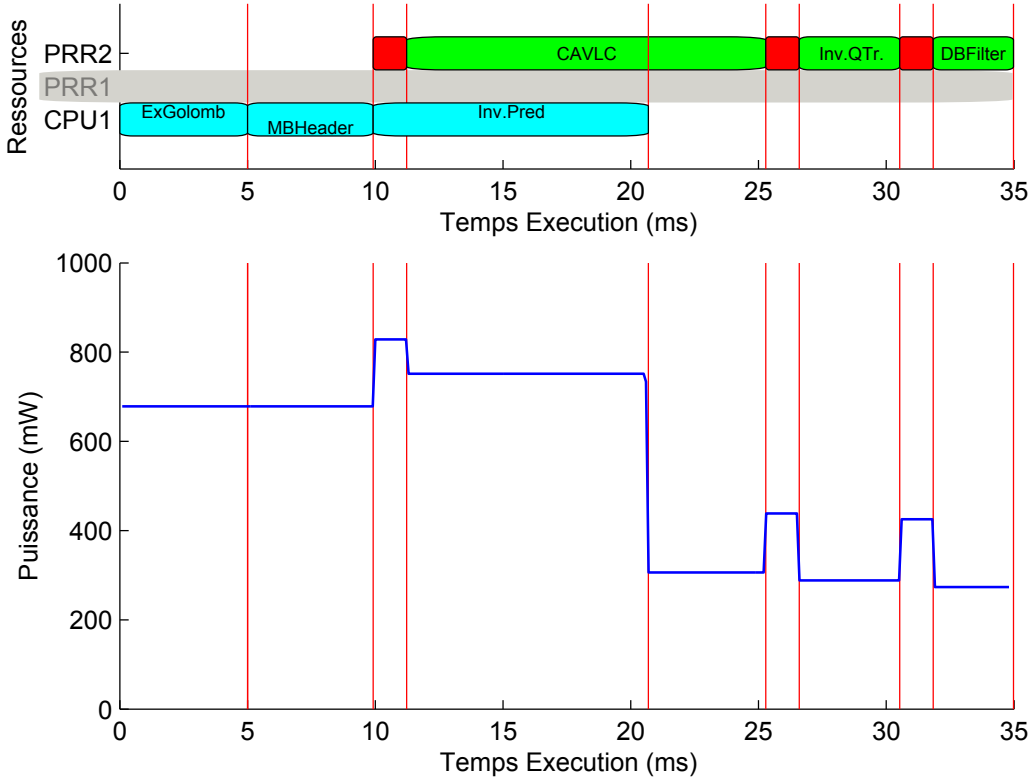

FIGURE 4.9: Implémentation et ordonnancement des tâches et profil de puissance pour la solution offrant l'énergie la plus faible.

Sur le diagramme de l'exécution des tâches, en haut des figures 4.9 et 4.10, on ne note pas de différence au niveau de l'ordonnancement. La seule différence se situe au niveau de l'implémentation : la tâche DBFilter est configurée sur la PRR1 pour la solution la plus rapide alors qu'elle est configurée sur la PRR2 pour la solution la moins consommatrice d'énergie. En effet le temps de configuration de la PRR2 est de 1.3 ms et l'exécution de Inv.QTr. prend 3.11 ms soit un total de 4.4 ms, alors que

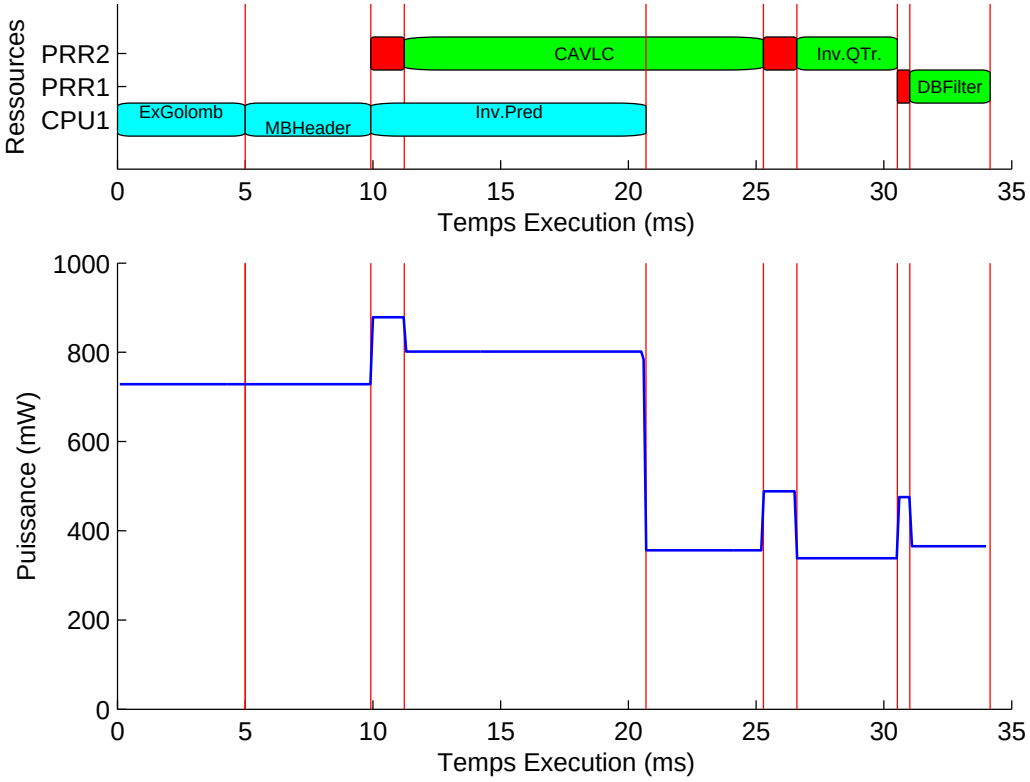

FIGURE 4.10: Implémentation et ordonnancement des tâches et profil de puissance pour la solution la plus rapide.

sur la PRR1, le temps de reconfiguration est de 0.5 ms et le temps d'exécution de 3.14 ms soit un total de 3.6 ms, ce qui est 0.8 ms plus rapide que sur la PRR2. En revanche, le fait de ne pas utiliser la ressource PRR1 permet de réduire sa consommation *idle* et permet d'envisager la suppression de la ressource du système et donc de réduire la consommation statique, ce qui est pris en compte par l'exploration. Le temps d'exécution obtenu est évidemment toujours plus élevé qu'une exécution matérielle statique à cause du surcoût de la reconfiguration (3 ms). Cependant avec ce contrôleur de reconfiguration optimisé, la consommation énergétique est environ $5mJ$ plus faible (-22%) que la version sans reconfiguration. Ces résultats montrent que l'utilisation des accélérateurs matériels statiques ne représente pas toujours une solution intéressante et que la reconfiguration permet de réduire la consommation d'énergie, à condition d'utiliser un contrôleur de reconfiguration efficace. Les valeurs sont indiquées dans le tableau 4.9. On remarque cependant que l'effacement (*blank*) n'est pas utilisé pour réduire la consommation. Les PRRs ne restent pas assez longtemps non utilisées et le coût énergétique de la reconfiguration pour effectuer l'effacement n'est pas amorti par l'économie de l'énergie *idle*.

Si on analyse l'occupation des ressources (haut des figures 4.9 et 4.10), on note que le taux d'utilisation est assez faible pour la solution la plus rapide, environ 10% pour

TABLE 4.9: Caractéristiques des résultats d'exploration

	Énergie (mJ)	Temps (ms)	Surface
Exécution logicielle (monoprocesseur)	47.91	87.92	1 CPU
Meilleur temps (reconfiguration dyn.)	20.94	34.16	1 CPU + 4400 slices
Meilleure énergie (reconfiguration dyn.)	19.45	34.97	1 CPU + 3200 slices
Exécution matérielle statique	25.10	31.11	1 CPU + 5189 slices

la PRR1 et 50% pour la PRR2 (les taux sont calculés uniquement avec l'exécution des tâches et sans compter le temps de reconfiguration dynamique). Les dépendances des tâches de l'application de décodage vidéo est à l'origine du faible taux d'utilisation des ressources. Dans cette application, la majorité des traitements de la vidéo sont effectués successivement sur des zones restreintes (macroblocs) de chaque image. Afin d'augmenter le parallélisme au niveau des tâches, il est envisageable de scinder en deux (ou plus) une partie du traitement vidéo et permettre l'augmentation du taux d'utilisation des ressources.

4.4.4 Résultats avec une version parallélisée de l'application

L'application est optimisée en exploitant le parallélisme d'une image. Dans un flux vidéo H.264, une image peut être décomposée en sous images (slices, ou tranches) sur lesquelles s'appliquent les processus de reconstruction d'une image. En d'autres termes, il est possible de traiter des moitiés (ou des quarts, huitième etc) d'image parallèle ce qui permet de réduire les temps d'exécution du même ordre. Les tâches de traitement du flux compressé (ExGolomb et MBHeader) ne peuvent pas être parallélisées, les autres tâches de traitement, elles, sont dupliquées pour fonctionner chacune sur la moitié de l'image. Le graphe de tâches obtenu est représenté sur la Figure 4.11. En réalité il y a un traitement supplémentaire de reconstruction de l'image à effectuer après le déblocage, l'influence de ce traitement est négligeable par rapport au reste de l'application et il n'est pas représenté sur le graphe.

L'exploration est effectuée à partir de ce graphe et le modèle de cette version parallélisée voit donc les temps d'exécution des tâches de reconstruction de l'image (CAVLC, Inv.QTr, Inv.Pred. et DBFilter) divisés par deux. Cette version de l'application est appelée H264_2 dans la suite en raison du dédoublement des slices contrairement à la première version, notée H264_1. La duplication des tâches est un point intéressant pour l'exploration et l'utilisation de la reconfiguration dynamique. En effet selon les contraintes de ressources et de temps, il est possible de n'effectuer qu'une seule reconfiguration pour exécuter les tâches des deux parties de l'image sur le même accélérateur.

Les résultats de l'exploration sont présentés à la Figure 4.12. On peut noter

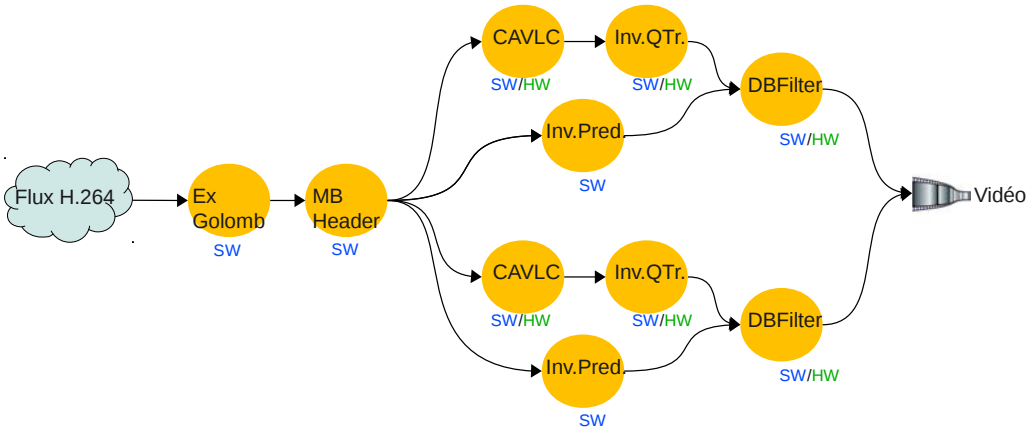

FIGURE 4.11: Graphe de tâches de l'application décodeur vidéo avec duplication des tâches de traitement.

la quantité importante de solutions explorées (244 000 solutions, contre 629 pour H264_1) puisque le nouveau graphe de tâches comporte quatre tâches supplémentaires ce qui complexifie énormément le nombre de possibilités d'ordonnancement et d'allocation. L'exploration prend trois heures sur un ordinateur de bureau (processeur *Intel Core2 Duo P8700-2.53GHz*) ce qui montre les limites de l'étude des solutions exhaustives.

La version dont l'exécution est entièrement logicielle prend exactement le même temps qu'avec la première description de l'application puisque il n'y a qu'un seul processeur et qu'il n'est pas possible de paralléliser les tâches. En revanche, les solutions exploitant la reconfiguration dynamique permettent de réduire le temps d'exécution ainsi que l'énergie par rapport à la précédente exploration.

Les solutions présentant le meilleur temps et la meilleure consommation énergétique sont extraites et sont identiques dans cet exemple. L'une d'elles est présentée dans la Figure 4.13. La solution retenue conserve la configuration de la PRR2 pour exécuter les deux instances de CAVLC évitant ainsi une reconfiguration. Le même cas de figure se présente pour la tâche DBFilter sur la PRR1. En revanche, en raison de la dépendance entre les tâches, il n'est pas possible de faire de même pour Inv.QTr. sans retarder la fin de l'exécution de l'application (qui aurait également un impact négatif sur l'énergie). Inv.QTr. est alors configurée en premier sur la PRR1 puis laisse la place à DBFilter. La seconde instance de Inv.QTr est exécutée sur la PRR2 après la fin de CAVLC. Grâce à l'augmentation du parallélisme du traitement de l'image, le temps d'exécution de l'application est plus faible, 30 ms soit 4 ms plus rapide que la version précédente. Malgré la réduction du temps d'exécution et l'augmentation du taux d'utilisation de la ressource PRR1 (taux de 20%), la consommation d'énergie est semblable à la version précédente, H264_1. Une reconfiguration supplémentaire ainsi qu'un laps de temps pendant lequel la PRR1 consomme de la puissance *idle*

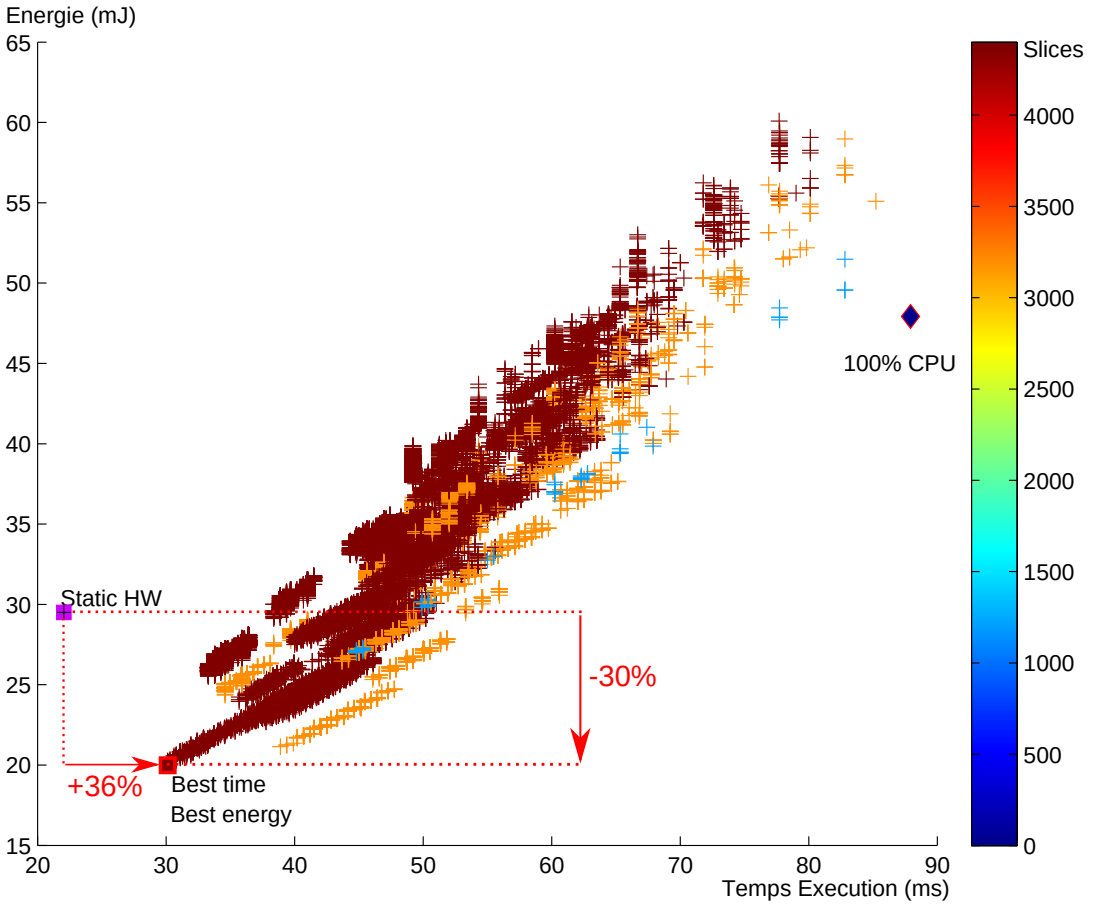

FIGURE 4.12: Ensemble des résultats obtenus en fonction des choix d'implémentation. Ces résultats sont présentés par leur énergie consommée en fonction du temps d'exécution et la surface matérielle requise est indiquée par une échelle de couleurs. Les résultats caractéristiques : exécution sur processeur uniquement (*100% CPU*), meilleur temps avec reconfiguration (*Best time*), plus faible énergie consommée avec reconfiguration (*Best energy*) et une exécution avec des accélérateurs matériels statiques (*Static HW*) sont présentés.

(entre 23 ms et 28 ms) engendre une augmentation de la consommation d'énergie. La réduction du temps d'exécution est liée à un meilleur taux d'utilisation des ressources matérielles. Ce n'est pas le cas avec les accélérateurs matériels statiques et l'énergie consommée par une implémentation statique des accélérateurs dans ce cas est plus importante à cause de la puissance statique et *idle* des accélérateurs. Les caractéristiques sont spécifiées dans le tableau 4.10 et en effet, la consommation énergétique de l'implémentation statique des accélérateurs passe de 25 à 30 mJ avec le dédoublement des accélérateurs matériels. Cette augmentation de la consommation, en dépit d'une accélération de près de 30%, est liée à la consommation statique plus importante qui résulte de l'augmentation du nombre d'accélérateurs. Une solution

logicielle biprocesseur est présentée pour comparaison. Le temps d'exécution et la consommation d'énergie sont supérieurs comparé aux solutions matérielles statiques et dynamiques.

Dans ces conditions d'utilisation, la reconfiguration dynamique permet de concurrencer une solution statique du point de vue de la consommation. En revanche le temps de reconfiguration augmente le temps d'exécution global. Les résultats d'ordonnancement spatio-temporel de la Figure 4.13 montrent également qu'un temps de reconfiguration peut être récupéré sur le temps d'exécution global. Dans la solution présentée, la reconfiguration de CAVLC est effectuée lorsque MBHeader est terminée. CAVLC ne peut commencer que lorsque sa configuration est terminée ce qui ajoute un délai pour l'exécution totale. La reconfiguration pourrait commencer avant la fin de l'exécution de la tâche précédente afin de pouvoir exécuter CAVLC aussitôt les dépendances satisfaites. Ceci est un point de l'exploration qui peut être amélioré.

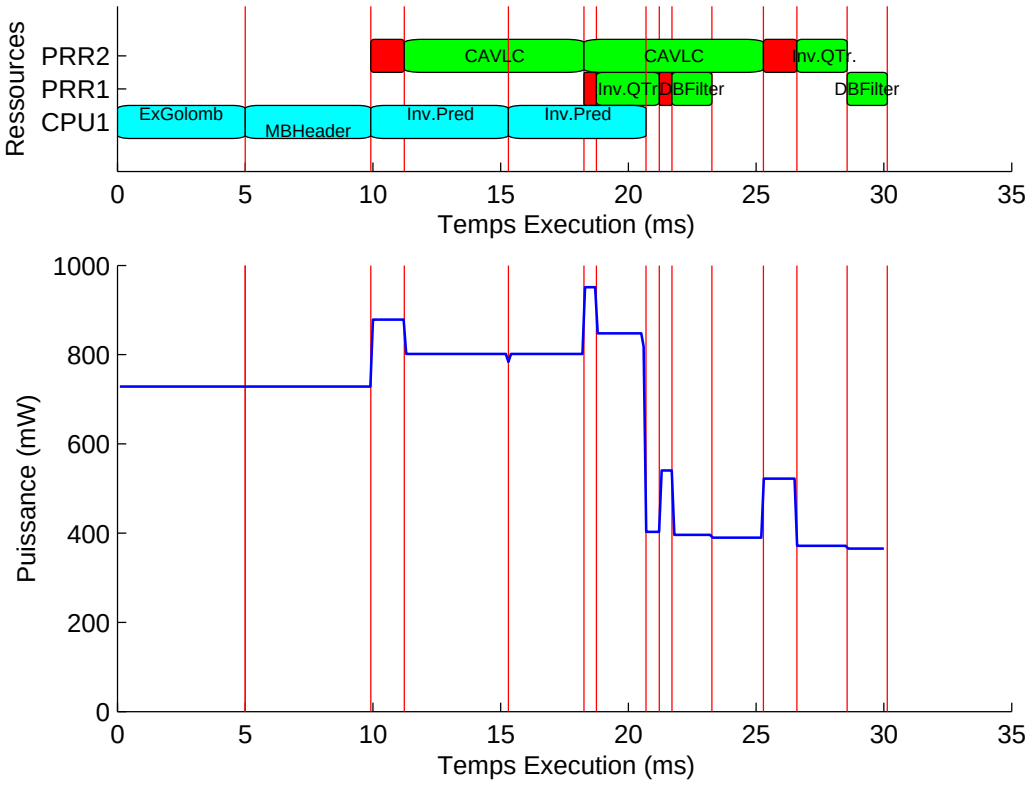

FIGURE 4.13: Implémentation et ordonnancement des tâches et profil de puissance pour la solution la plus rapide.

Les explorations sur un cas d'exemple d'une application de décodage vidéo ont montré que la reconfiguration a bien un impact sur la consommation globale d'un système. La puissance nécessaire pour effectuer la reconfiguration peut provoquer

TABLE 4.10: Caractéristiques des résultats d'exploration

	Énergie (mJ)	Temps (ms)	Surface
Exécution logicielle (monoprocesseur)	64.05	87.92	1 CPU
Exécution logicielle (biprocesseur)	48.90	48.92	2 CPUs
Meilleure énergie (reconfiguration dyn.)	19.99	30.13	1 CPU + 4400 slices
Exécution matérielle statique (H264_2)	29.5	22.05	1 CPU + 10378 slices
Exécution matérielle statique (H264_1)	25.10	31.11	1 CPU + 5189 slices

des pics de puissance à prendre en compte, cependant lorsque la reconfiguration est rapide, elle permet d'augmenter le taux d'utilisation des PRRs avec un impact temporel faible. L'augmentation du taux d'utilisation de ces ressources permet alors de diminuer la surface totale du système pour le même traitement effectué. La diminution de la surface permet de réduire la consommation statique et *idle*. Malgré un coût lié à la reconfiguration dynamique, la consommation globale du système est réduite.

4.5 Modélisation et exploration AADL

Pour faciliter son utilisation, PREexplorer repose sur une modélisation existante pour éviter des étapes supplémentaires lors de la conception d'un système. PREexplorer s'appuie sur la modélisation AADL du projet OpenPEOPLE. Les paramètres nécessaires à l'exploration ainsi que les résultats sont transcrits pour être compatible avec AADL. Ainsi les étapes de modélisation de l'application et du système ne sont pas un travail supplémentaire à d'autres modélisations existantes. Cependant AADL a dû être étendu pour permettre la modélisation des FPGAs et pour supporter la description de la reconfiguration dynamique partielle.

4.5.1 Extension de la modélisation pour les FPGAs

AADL est initialement destiné à décrire les systèmes à base de processeurs qu'ils soient monoprocesseur, multiprocesseurs homogènes ou hétérogènes. Les mémoires peuvent également être modélisées et le mode de fonctionnement peut être choisi. Les applications peuvent également être modélisées et caractérisées afin de permettre des analyses et vérifications (puissance maximale, charge des processeurs, contraintes temps réel...). Cependant lorsque l'architecture contient une ressource reconfigurable, le méta modèle AADL ne permet peut pas d'être utilisé directement. En effet, les propriétés de ces ressources sont différentes de celles d'un processeur et en particulier les notions de surface et de reconfigurabilité sont spécifiques.

Pour permettre la modélisation des systèmes intégrant une ressource reconfigu-

rable, le langage est adapté pour la description de ces architectures. L'approche proposée est de modéliser un FPGA à trois niveaux d'abstraction [2], représentés Figure 4.14. Le premier niveau considère le FPGA comme un système contenant une région statique et une région reconfigurable. À ce niveau, la description est indépendante de la technologie et est générique. Le second niveau spécifie la partie statique du FPGA et les ressources disponibles (CLBs, DSPs, mémoires...) dans la région reconfigurable, le contenu du FPGA est maintenant fixé. Finalement le troisième niveau concerne la configuration de la ressource reconfigurable. Les blocs implémentables dans cette ressource sont décrits et les propriétés spécifiques (surface et puissance notamment) sont précisées. La modélisation permet de décrire les architectures reconfigurables et les accélérateurs matériels qui y sont configurés. Cependant la reconfiguration dynamique n'est pas supportée par cette description.

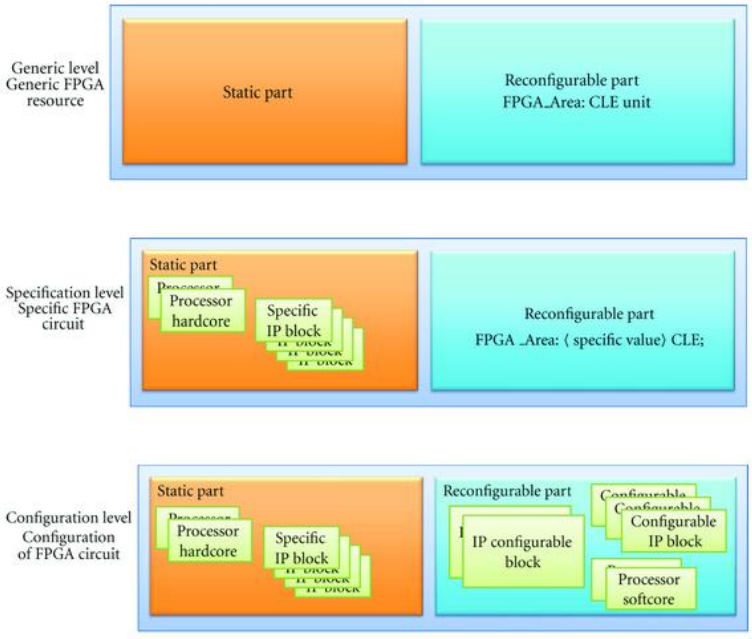

FIGURE 4.14: Approche pour la modélisation d'un FPGA en AADL [2].

4.5.2 Modélisation de la reconfiguration dynamique

Afin de pouvoir modéliser les changements de configurations, les *modes* proposés par AADL peuvent être utilisés. Les *modes* permettent de décrire l'état du système en fonction d'évènements. Les *modes* permettent d'apporter une dynamicité à la description AADL, ainsi les changements d'états d'un processeur (fréquence de fonctionnement ou mode de repos par exemple) peuvent être modélisés. De même, un bloc peut être instancié dans certains *modes* uniquement. Nous utilisons les *modes*

pour décrire quels sont les blocs instanciés dans la(les) région(s) reconfigurable(s) et des propriétés spécifiques peuvent être renseignées. La reconfiguration dynamique est alors représentée par un changement de *mode* et la représentation d'une exécution de plusieurs blocs est donnée par une succession de transitions de *modes*. Des propriétés relatives au changement de *mode* sont spécifiées et correspondent aux caractéristiques de la reconfiguration, notamment le temps et l'énergie nécessaires. Une solution d'exécution d'une application est représentée par une succession ordonnée de transitions de *modes* qui décrivent les reconfigurations des accélérateurs matériels. Cette approche est représentée Figure 4.15. Les transitions de *modes* sont déterminées à partir des résultats de PREexplorer. Ainsi les choix d'implémentation effectués lors de l'exploration sont modélisés et il est possible de poursuivre la modélisation et la conception du système avec AADL.

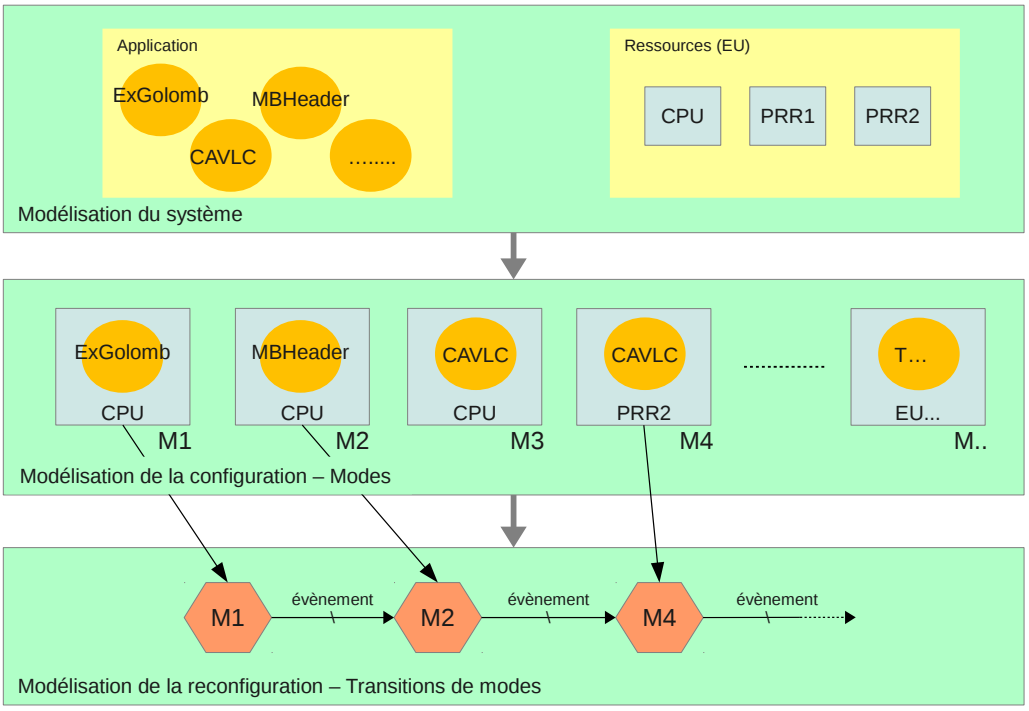

FIGURE 4.15: Approche pour la modélisation de la reconfiguration dynamique en AADL.

4.5.3 Intégration de l'exploration dans l'environnement AADL

Afin de pouvoir être utilisé facilement, PREexplorer est couplé au flot de modélisation AADL. La représentation graphique de ce flot est donnée Figure 4.16. La description du système et de l'application est effectuée de manière classique en utilisant AADL. L'ensemble des paramètres, présentés dans ce chapitre à la section 4.2, doit être renseignés afin de permettre l'exploration d'un système reconfigurable dy-

namiquement. Lorsque ces paramètres sont spécifiés dans la description AADL, un script traduit les paramètres depuis la modélisation en langage AADL vers un fichier de paramètres en langage *Matlab* ou *Octave* (le langage de fonctionnement actuel de PREexplorer). L'exécution de PREexplorer est lancée et l'ensemble des solutions respectant les contraintes d'implémentation du système sont proposées à l'utilisateur. L'ordonnancement et les choix d'allocation de la solution retenue sont extraits et convertis par un script (en cours de développement) vers le flot de modélisation AADL, en utilisant les transitions de *modes*. Le retour des résultats de PREexplorer dans la description AADL permet de poursuivre le flot de modélisation et de conception avec les autres outils disponibles pour ce standard. L'intégration de STORM [93, 94], un outil permettant l'exploration de l'ordonnancement multiprocesseur des tâches d'une application aux contraintes temps réel, est également prévue afin de compléter l'exploration de PREexplorer au niveau des politiques d'ordonnancement et d'économie d'énergie pour les processeurs embarqués. La(les) solution(s) pour l'exécution de l'application retenue(s) peut(peuvent) ensuite être raffinée(s) via le flot de conception postérieur à la modélisation AADL. Les contraintes de temps d'exécution et d'énergie (en particulier) sont vérifiées et si les contraintes ne sont pas respectées, la description du système ou de la plateforme doit être modifiée et l'exploration réitérée.

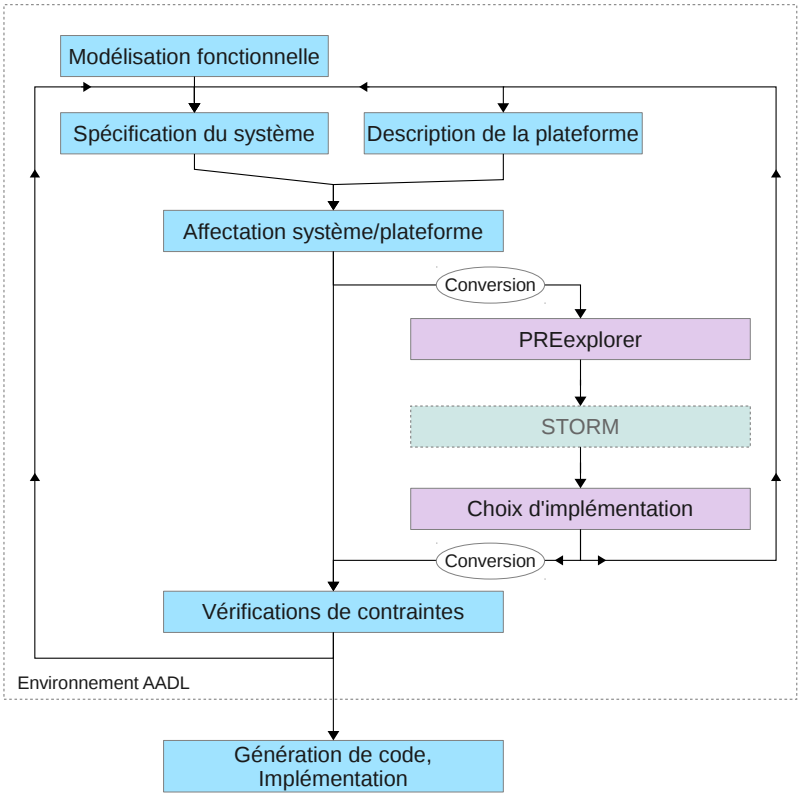

FIGURE 4.16: Flot d'exploration complet.

L'automatisation de l'extraction des paramètres, le choix des résultats à conserver et l'injection des résultats d'exploration sont encore des travaux en cours. Le travail d'adaptation du méta modèle AADL pour modéliser la reconfiguration ainsi que l'intégration de PREexplorer dans le flot ont été effectués principalement grâce aux travaux de stage en rapport avec cette thèse par Ferhat Abbas, Rim Abid, Houssem Hamoud et Hanen Khlif.

4.6 Conclusion

La reconfiguration dynamique partielle doit être rapide par rapport à l'exécution des tâches pour que son utilisation ait un impact temporel faible au niveau de l'application. Un contrôleur de reconfiguration optimisé est proposé, il permet d'obtenir des débits très élevés, jusqu'à 1.4 GB/s, avec une consommation maîtrisée pour favoriser l'utilisation de la reconfiguration.

Un algorithme d'exploration de l'implémentation des tâches pour les systèmes reconfigurables et multiprocesseurs hétérogènes est présenté. Le but de cette exploration est d'obtenir un ordonnancement et un choix d'allocation et d'implémentation des tâches qui permettent de minimiser par exemple le temps d'exécution ou l'énergie en fonction des paramètres et contraintes du système modélisées.

L'exploration permet d'analyser, lors de l'étape de conception, l'impact de la reconfiguration dynamique partielle sur une application de décodage vidéo en exemple. Deux modèles de la consommation de la reconfiguration dynamique, présentés dans le chapitre précédent, sont testés et les résultats montrent que le niveau de détail du modèle à grain fin estime un pic de puissance 9% plus élevé que le modèle à gros grain. Ces pics de puissance provoqués par la reconfiguration doivent être pris en compte dans les systèmes sensibles. Sur le même cas d'exemple, l'exploration montre que la reconfiguration dynamique partielle des accélérateurs matériels peut être utilisée pour réduire la consommation d'énergie. Grâce à la reconfiguration dynamique, une réduction énergétique de 58% par rapport à une exécution logicielle monoprocesseur est estimée. Comparée à une exécution matérielle statique, la consommation énergétique de la solution reconfigurable est 22% plus faible avec un délai supplémentaire d'exécution de 10 à 36% et une réduction de la surface de 15 à 57%.

Finalement, la modélisation du système nécessaire à l'exploration est apportée par AADL. Le langage de modélisation est adapté pour supporter la description des ressources reconfigurables dynamiquement et l'algorithme est couplé au flot de modélisation AADL ce qui facilite son utilisation. Les résultats de l'exploration sont exportés en AADL pour permettre la poursuite du flot de conception habituel.

Conclusion et Perspectives

Conclusion

Les évolutions technologiques des systèmes sur puce et la complexité croissante des applications qui les accompagne posent des problèmes de conception qui nécessitent de nouvelles approches. L'augmentation permanente des besoins en puissance de calcul et la progression de la technologie repoussent toujours plus les limites de l'intégration des circuits électroniques et la consommation est un problème majeur. La dissipation, le temps de fonctionnement sans panne (MTTF), l'autonomie et les préoccupations environnementales sont autant de contraintes qui nécessitent la maîtrise de l'efficacité énergétique et celle-ci ne se limite plus désormais au seul domaine de l'embarqué. Dans ce contexte, ces travaux de thèse ont permis de définir une caractérisation très complète de la consommation des architectures hétérogènes reconfigurables qui englobe les aspects liés aux techniques de reconfiguration dynamique partielle, mais également les autres unités logicielles. À partir de cette modélisation, une méthode pour l'exploration avec estimation systématique de la consommation a pu être proposée et les études de validation montrent des résultats exploitables pour le niveau d'abstraction d'entrée (système). Les résultats permettent par exemple de déterminer si l'emploi de la reconfiguration dynamique est plus intéressant qu'une solution statique ou logicielle, en proposant une quantification des gains, à la fois en performance, en consommation et en surface.

Nous répondons alors à une première interrogation : grâce à la réutilisation des ressources, la reconfiguration dynamique partielle permet-elle d'utiliser la surface libérée pour augmenter l'efficacité des accélérateurs matériels en exploitant de parallélisme ?

Les mesures menées montrent, sur les exemples utilisés, que la variation du niveau de parallélisme fait apparaître des réductions énergétiques jusqu'à un facteur de 2.26. Ce gain est obtenu principalement grâce à la réduction du temps d'exécution qui entraîne linéairement une baisse de la consommation d'énergie. Cependant cette première réponse est incomplète ; l'exploitation du parallélisme d'un accélérateur matériel accroît sa surface, ce qui augmente la taille du bitstream nécessaire à la configuration. Le temps de reconfiguration est alors allongé et il a une influence

sur la consommation de la reconfiguration de cet accélérateur.

Quel est l'impact énergétique de la reconfiguration dynamique et comment garantir une meilleure efficacité énergétique dans les systèmes reconfigurables ?

Un modèle de consommation précis de la reconfiguration est nécessaire pour estimer avec pertinence l'intérêt de la reconfiguration d'un accélérateur matériel en fonction de son niveau de parallélisme. Pour obtenir ce modèle, la consommation du cœur d'un FPGA (Xilinx Virtex-5) lors de la reconfiguration a été mesurée. L'analyse détaillée a permis de définir trois modèles de consommation à différents niveaux de granularité. La précision énergétique des modèles atteint 98% pour le plus fin.

Les modèles développés sur les deux aspects précédents permettent de répondre à la problématique initiale par l'utilisation d'une méthode d'exploration. Cette dernière est supportée par l'outil proposé, PREexplorer, qui prend ainsi en compte l'ensemble des coûts énergétiques d'un système pour évaluer quantitativement et avec précision la question de l'utilisation de la reconfiguration dynamique. Les options d'allocation des tâches d'une application sur des processeurs ou sur des accélérateurs matériels sont analysées en tenant compte de la reconfiguration dynamique. Les résultats de l'exploration sur une application de décodage vidéo (H.264) montrent que l'utilisation de la reconfiguration dynamique, lorsqu'elle est suffisamment rapide et employée à bon escient, permet de réduire la consommation énergétique de 60% par rapport à une exécution sur processeur, et elle est également réduite de 35% comparée à un système utilisant des accélérateurs matériels implémentés statiquement. Cette réduction de la consommation énergétique est obtenue principalement grâce à la diminution de la surface nécessaire pour les accélérateurs matériels, jusqu'à 57%, allégeant ainsi la consommation statique et dynamique au repos (*idle*).

Le recours régulier à la reconfiguration dynamique pour le placement des tâches sous contraintes énergétiques est une opération complexe et délicate qui doit pouvoir être analysé dans les phases amont des flots de conception. La méthode d'exploration proposée permet cette analyse avec toutefois des points à améliorer. L'ordre d'exécution des tâches lors de l'exploration est dicté par un ordonnancement simple : une tâche est exécutée dès lors que ses dépendances sont satisfaites. Il est possible d'obtenir d'autres résultats, peut-être meilleurs, avec une politique d'ordonnancement différente. L'étude de l'ordonnancement augmente l'espace d'exploration et ces travaux se focalisent sur l'utilisation de la reconfiguration. L'espace d'exploration de l'implémentation et l'allocation est déjà important, la complexité de l'algorithme est exponentielle et l'étude pour des applications complexes est longue. La proposition de formalisation et de modélisation de l'utilisation de la reconfiguration dynamique

a montré que de nombreux paramètres influencent la consommation. La minimisation de l'énergie est un point difficile dans ce cas car les choix d'implémentation pris à chaque instant ont un impact important sur l'exécution des tâches suivantes. Nous avons donc choisi une exploration exhaustive des possibilités de reconfiguration pour proposer les solutions qui minimisent la consommation ou le temps d'exécution. Des améliorations aux méthodes proposées dans ce mémoire sont présentées dans la section suivante, avec une orientation pour la suite de ces travaux de thèse.

Perspectives

Les perspectives de ce travail portent en premier lieu sur des propositions architecturales pour favoriser l'utilisation de la reconfiguration lors de la conception de circuits reconfigurables. Un autre point concerne la construction des modèles de consommation de la reconfiguration et leur extension à d'autres circuits reconfigurables. La troisième partie évoque les améliorations à apporter à l'exploration, en particulier la prise en charge d'autres politiques d'ordonnancement, le partitionnement des PRRs et l'extension à des architectures hétérogènes plus diversifiées. Finalement, la réduction de la complexité pour la prise de choix d'allocation en ligne est abordée pour, par exemple, être supportée par un service de système d'exploitation.

Conception de circuits reconfigurables. L'analyse de la puissance consommée pendant la reconfiguration montre des surconsommations (section 3.3.3.1) qui sont peut-être liées à des courts-circuits lors des changements de la configuration des interconnexions. Afin de valider cette hypothèse, il faut mener des expériences sur le comportement de la puissance statique lors de la reconfiguration. Si celle-ci est validée, afin de réduire la consommation, il serait intéressant soit de concevoir un circuit reconfigurable dont les interconnexions peuvent éviter de provoquer des courts-circuits.

Pour éviter ces courts-circuits, la première possibilité consiste à mettre en haut impédance les sorties (sorties 3 états) des LUTs et bascules de l'ensemble de la région reconfigurée. Cette méthode risque d'augmenter le délai des interconnexions et une difficulté réside dans la sélection de la région complète avant la reconfiguration. La seconde possibilité est de proposer un effacement complet rapide de la PRR, cette méthode est détaillée ci-dessous. La reconfiguration s'effectuerait en deux étapes, la première un effacement complet de la PRR, puis une seconde étape de configuration du nouvel accélérateur. Le coût énergétique de l'effacement est à contrebalancer avec le gain obtenu en évitant les courts-circuits. L'étape d'effacement doit être très rapide.

De plus, l'exploration montre que l'effacement de la PRR n'est utilisé dans aucun des exemples traités car le coût énergétique de la reconfiguration d'une tâche *blank* est plus important que l'énergie qu'elle permet d'économiser. Il serait probablement intéressant de considérer une architecture matérielle permettant de réaliser l'effacement d'une PRR en quelques cycles d'horloge, afin d'augmenter le potentiel réduction de ces techniques.

Par exemple, les FPGAs Virtex 5 et 6 de chez Xilinx ont un découpage des régions reconfigurables en *frames*. Chacun de ces blocs nécessite 41 mots de configuration (pour des *frames* CLBs). En conservant cette architecture, l'utilisation d'un mot spécifique pour la commande d'effacement d'une *frame* permettrait de réduire le temps d'effacement (et donc l'énergie correspondante) par 41. Un exemple de contenu du *bitstream* est présenté Figure 4.17 et l'architecture associée, Figure 4.18. Le contenu du bitstream est chargé par un bus de configuration. Sur ce bus, un bloc de décodage d'adresses est présent pour permettre la sélection de la bonne zone mémoire de la configuration. Une détection de mot d'effacement est ajoutée à ce bloc. Ce mot provoque une remise à zéro de la mémoire de configuration et effectue ainsi l'effacement en une seule commande par *frame*.

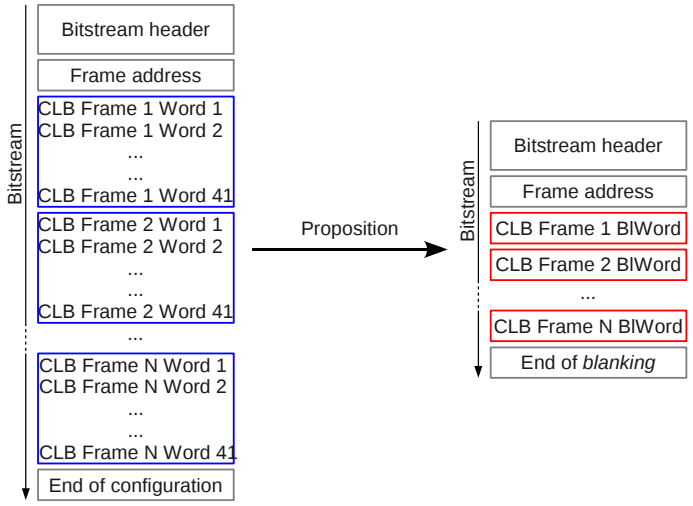

FIGURE 4.17: Contenu du bitstream d'effacement avec l'utilisation d'un mot spécifique (BlWord).

L'étude de la reconfiguration montre également qu'un débit de reconfiguration important est primordial pour que cette technique soit utilisée dans des applications avec de fortes contraintes de temps (temps réel). L'efficacité énergétique de la reconfiguration est également importante pour que son impact soit faible sur la consommation globale du système. Pour améliorer ces caractéristiques, le contrôleur de reconfiguration pourrait être intégré à l'interface de reconfiguration du circuit par

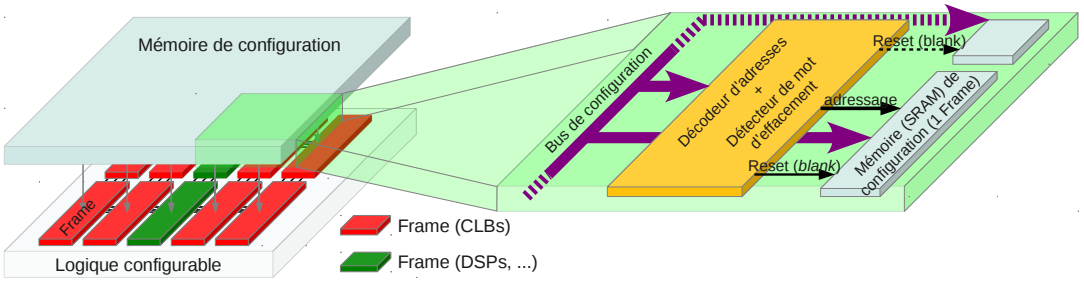

FIGURE 4.18: Architecture de la mémoire de reconfiguration pour l'utilisation d'un mot spécifique d'effacement.

une ressource dédiée. Ce bloc permet de s'affranchir de la consommation et de la latence de la matrice d'interconnexion. Il fonctionne selon le principe d'un DMA : il aurait un accès direct à la mémoire de reconfiguration et à la mémoire externe de stockage des bitstreams, grâce à un bus haute performance. Le contrôle de la reconfiguration est alors effectué par des pointeurs accessibles depuis l'intérieur du circuit. Selon cette architecture, représentée Figure 4.19, les délais de reconfiguration sont optimisés matériellement et l'efficacité énergétique meilleure grâce à l'utilisation d'un bloc matériel dédié.

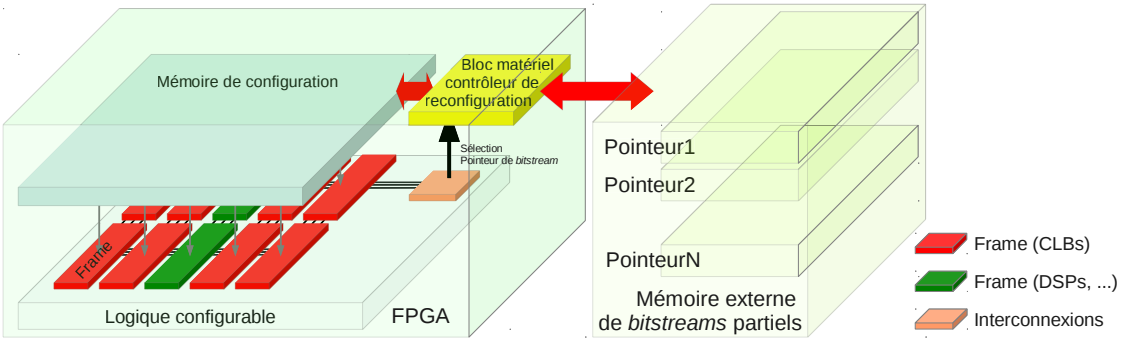

FIGURE 4.19: Architecture intégrant une ressource matérielle pour le contrôle de la reconfiguration.

Modèles de consommation. La construction du modèle à grain fin de la consommation de la reconfiguration dynamique (section 3.3.4) est complexe à effectuer. Elle nécessite une bonne connaissance de l'architecture et une étape de caractérisation avec plusieurs tâches différentes à reconfigurer. De plus, ce modèle a quelques déviations par rapport aux mesures. Une possibilité d'amélioration est d'avoir une connaissance plus précise du rôle de chaque bit des données de configuration, ce qui est difficile actuellement dans les FPGAs étudiés (Xilinx Virtex-5 ou Virtex-6). La connaissance précise du rôle des données de configuration permettrait de simplifier le modèle par la diminution du nombre de calculs de la distance de Hamming en la

portant uniquement sur les bits/mots ayant la plus forte influence sur la consommation. Les bits à prendre en compte sont particulièrement ceux qui provoquent les éventuels courts-circuits (cette hypothèse est à valider et présentée dans le point précédent) et ceux ayant une influence sur la puissance *idle* (propagation d'horloge et activation de blocs particuliers).

Par ailleurs, la validation de ces modèles doit être poursuivie sur d'autres circuits, notamment sur le Virtex-7 et d'autres types d'architectures reconfigurables, ce qui pourra également fournir des pistes sur la généralisation de la construction des modèles en limitant les étapes complexes de mesures.

Amélioration de l'exploration. Pour réduire la complexité, l'exploration se base sur un ordonnancement simple. Il existe certainement un potentiel important d'amélioration des solutions par un ordonnancement plus optimisé. La contrepartie est bien sûr la taille de l'espace exploré puisqu'il s'agit d'un compromis. Il est certainement possible de combiner l'utilisation d'heuristiques de recherches pour aboutir à un ordonnancement plus adapté à la réduction de la consommation.

Sans nécessairement proposer une exploration pour un système préemptif, d'autres politiques d'ordonnancement pourraient être utilisées ou proposées. Par exemple, il n'est pas nécessairement intéressant d'exécuter une tâche dès que ses dépendances le permettent, mais attendre (jusqu'à un certain point ne retardant pas l'exécution globale) peut aider à maintenir la ressource matérielle effacée et réduire la consommation. D'un autre côté, la reconfiguration peut être opérée en avance, autorisant une exécution plus tôt, dès que les dépendances sont satisfaites. Une continuité de ces travaux consisterait à étudier des extensions permettant l'étude de l'allocation et de l'ordonnancement selon différentes politiques, tenant compte de la consommation énergétique, sur des ressources hétérogènes logicielles et matérielles.

L'état de l'art mentionne que le partitionnement de la ressource reconfigurable en régions peut accroître la quantité de ressources non utilisées en raison : i) du grain de reconfigurabilité qui nécessite un arrondi supérieur aux besoin des tâches, et ii) des choix de dimensionnement effectués par le concepteur, par exemple de petites tâches peuvent être implémentées sur des régions inutilement larges. Pour améliorer l'utilisation de la reconfiguration, l'exploration pourrait être couplée à un outil de partitionnement des ressources reconfigurables. Cette étape étant actuellement effectuée par l'utilisateur, un partitionnement automatique du FPGA en PRRs faciliterait l'exploration. De plus, les résultats énergétiques ou temporels pourraient être améliorés grâce à un découpage véritablement adapté aux tâches de l'application.

Actuellement, la seule technique de réduction de la consommation utilisée explici-

tement est l'effacement d'une région. L'état de l'art mentionne d'autres techniques dont le *clock gating* et les changements de fréquence, principalement, qui peuvent être utilisées dans les circuits reconfigurables. La prise en compte de ces techniques permettrait d'élargir l'espace d'exploration et de compléter l'effacement pour réduire la consommation.

La modélisation est destinée aux architectures reconfigurables et aux unités logicielles. La méthodologie pourrait être étendue pour prendre en compte d'autres ressources des systèmes hétérogènes, par exemple les mémoires, d'autres accélérateurs tels que les GPUs etc. et les aspects de communication pour aboutir à une méthode globale et complète couvrant un grand nombre d'architectures. En rapport avec la consommation, l'extraction des informations sur les points chauds du circuit permettrait d'étudier les cycles de variation de la température et de considérer la durée de vie du circuit (MTTF). Ces aspects ouvrent la voie pour une meilleure gestion des phénomènes de type *dark silicon*, où toutes les parties du circuit ne peuvent fonctionner en même temps pour des raisons de densité de puissance sur les puces.

Réduction de la complexité. La complexité exponentielle de l'algorithme d'exploration et le nombre exhaustif de solutions analysées sont rapidement un problème dans des applications réalistes. Ce problème touche ici à la conception mais peut se poser également si l'on souhaite effectuer les choix d'implémentation *in situ*, par exemple par un service intégré dans un système d'exploitation embarqué.

Une piste intéressante peut être fournie par l'étude de méthodes de résolutions algébriques qui s'appuient sur les formalisations telles que celles exprimées au chapitre 2.5. La minimisation de la consommation par résolution algébrique, si elle est possible, risque d'être complexe. La définition d'heuristiques, par des choix d'allocation simplifiés, est une possibilité à envisager pour réduire la complexité de résolution. Le risque d'utiliser des heuristiques est de trouver un minimum local et d'obtenir un résultat éloigné du minimum global pour le paramètre considéré (consommation ou temps d'exécution en particulier). L'algorithme d'exploration permettra de vérifier la pertinence de ces heuristiques.

Une autre piste de recherche est l'utilisation de la théorie des jeux pour représenter ce problème. Selon la définition, la théorie des jeux est un ensemble d'outils qui permet d'étudier des situations où des individus prennent des décisions en fonction ou en anticipation des choix des autres. L'objectif est de déterminer une stratégie pour chaque individu qui permette d'aboutir à un résultat final optimal.

La théorie des jeux est un domaine vaste, qui semble se prêter à notre problématique. Par exemple, si l'on considère qu'un "individu" est une tâche, les "décisions"

dont il dispose sont les choix d'implémentation et d'allocation et le "résultat final" est l'optimisation de la consommation. Cette approche est schématisée Figure 4.20. Après avoir caractérisé ces éléments et spécifié le type de jeu, *a priori* coopératif (les choix d'implémentation et d'allocation effectués pour les premières tâches sont connus par les tâches suivantes), les outils d'analyse existants pourraient aider à la résolution de la problématique de la réduction de la consommation dans les circuits hétérogènes reconfigurables. Grâce aux outils et méthodes associés, la résolution du jeu est probablement moins complexe qu'une exploration exhaustive.

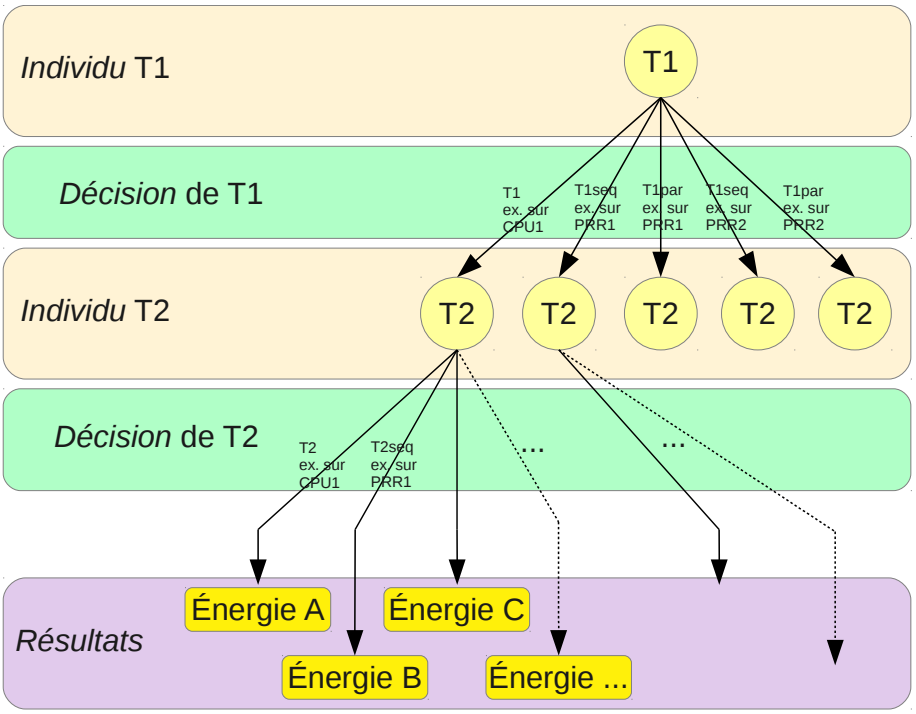

FIGURE 4.20: Représentation des choix d'implémentation et d'allocation sous forme d'arbre pour l'approche basée sur la théorie des jeux

Publications

- Bonamy, R. ; Chillet, D. ; Sentieys, O. ; Bilavarn, S., "Power consumption model for partial dynamic reconfiguration". *In Proc. of International Conference on ReConFigurable Computing and FPGA (RECONFIG'2012)*, December 2012.

- Bonamy, R. ; Hung-Manh Pham ; Pillement, Sebastien ; Chillet, D., "UPaRC-Ultra-fast Power-aware Reconfiguration Controller", *In Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp.1373,1378, 12-16 March 2012

- Bonamy, R. ; Chillet, D. ; Sentieys, O. ; Bilavarn, S., "Towards a power and energy efficient use of partial dynamic reconfiguration", *6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pp.1,4, 20-22 June 2011

- Bonamy, R. ; Chillet, D. ; Sentieys, O. ; Bilavarn, S., "Parallelism level impact on energy consumption in reconfigurable devices". *ACM SIGARCH Computer Architecture News*, vol.39(4) :104–105, December 2011.

- Blouin, D. ; Chillet, D. ; Senn, E. ; Bilavarn, S. ; Bonamy, R. ; C.Samoyeau. "AADL extension to model classical FPGA and FPGA embedded within a SoC". *International Journal of Reconfigurable Computing*,(Article ID 425401) :15 pages, 2011.

- Blouin, D. ; Chillet, D. ; Senn, E. ; Bilavarn, S. ; Bonamy, R. ; C.Samoyeau. "FPGA modeling for SoC design exploration". *In International Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART)*, London, May 2011.

Glossaire et liste des acronymes

AADL *Architecture Analysis & Design Language*, langage de modélisation.

ASIC *Application Specific Integrated Circuit*.

b*itstream* données de configuration d'un FPGA.

b*lank* configuration permettant d'effacer une PRR.

BRAM *Block RAM*, mémoire interne aux FPGAs.

CLB *Configurable Logic Block*.

CMOS *Complementary Metal-Oxide-Semiconductor*, technologie de fabrication des circuits intégrés.

DMA *Direct Memory Access*, accès direct à la mémoire.

DPR *Dynamic and Partial Reconfiguration*, reconfiguration dynamique et partielle.

DSP *Digital Signal Processor*, processeur spécialisé dans le traitement du signal, dans le contexte des FPGAs, ces DSPs sont généralement des blocs multiplieurs rapides.

DVFS *Dynamic Voltage and Frequency Scaling*.

EDK *Embedded Development Kit*, suite d'outils proposée par Xilinx pour la conception des sytèmes à base de FPGA.

EU *Execution Unit*, unité d'exécution (ressource).

F*rame* bloc élémentaire reconfigurable dynamiquement.

FPGA *Field-Programmable Gate Array*, Circuit reconfigurable.

FSM *Finite State Machine*, machine d'états.

i*dle* État d'inactivité d'une ressource.

ICAP *Internal Configuration Access Port*, port interne des FPGAs pour permettre la reconfiguration dynamique.

LUI *Loop Unrolling Index*, indice de déroulage de boucle.

LUTs *Look Up Tables*, Tables de vérité.

M*icroBlaze* Microprocesseur (Xilinx) configuré dans le FPGA, cœur logiciel.

OEL *Ordered Execution List*, liste d'exécution ordonnée.

OpenPEOPLE *Open-Power and Energy Optimization PLatform and Estimator*, projet de recherche porté par l'ANR.

PlanAhead Outil Xilinx utilisé principalement pour définir les PRRs et générer les bitstreams partiels.

PLB *Processor Local Bus*, bus du processeur *MicroBlaze*.

PREexplorer Outil d'exploration de l'implémentation et de l'allocation des tâches et ressources pour la consommation d'énergie dans les architectures hétérogènes reconfigurables.

PRR *Partial Reconfigurable Region*, région reconfigurable partiellement et dynamiquement.

SoC *System-on-Chip*, système sur puce.

SRAM *Static Random Access Memory*, type de mémoire vive.

UPaRC *Ultra-fast Power-aware Reconfiguration Controller*, contrôleur de reconfiguration optimisé.

UReC *Ultra-fast Reconfiguration Controller*.

Virtex Famille de FPGA développés par Xilinx destinés au calcul de haute performance.

x*ps_hwicap* Contrôleur de reconfiguration fourni par Xilinx.

Xilinx Entreprise de semiconducteurs, développant principalement des FPGAs.

Bibliographie

[1] Andres Garcia, Wayne Burleson, and Jean-Luc Danger. Power Modelling in Field Programmable Gate Arrays (FPGA). In *International Workshop on Field Programmable Logic and Applications*, pages 396–404. Springer, 1999. URL http://www.springerlink.com/content/xl39fkjyqepbv4fd.

[2] Dominique Blouin, Daniel Chillet, Eric Senn, Sébastien Bilavarn, Robin Bonamy, and Christian Samoyeau. AADL Extension to Model Classical FPGA and FPGA Embedded within a SoC. *International Journal of Reconfigurable Computing*, vol. 2011 :Article ID 425401, 2011.

[3] Anantha P Chandrakasan and Robert W Brodersen. *Low power digital CMOS design*. Kluwer Academic Pub, 1995.

[4] G.E. Moore. Progress in digital integrated electronics. In *Electron Devices Meeting, 1975 International*, volume 21, pages 11 – 13, 1975.

[5] Intel Inc. Intel 22nm 3-D Tri-Gate Transistor Technology. Technical report, 2011.

[6] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current : Moore's law meets static power. *Computer*, 36(12) :68 – 75, dec. 2003. ISSN 0018-9162. doi : 10.1109/MC.2003.1250885.

[7] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage : A temperature-aware model of subthreshold and gate leakage for architects. Technical report, 2003.

[8] Sung Mo Kang. Accurate simulation of power dissipation in vlsi circuits. *IEEE Journal of Solid-State Circuits*, 21(5) :889–891, 1986.

[9] F.N. Najm. A survey of power estimation techniques in vlsi circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4) :446–455, 1994. ISSN 1063-8210.

[10] Anand Raghunathan, Sujit Dey, and Niraj K. Jha. Register-transfer level estimation techniques for switching activity and power consumption. In *Proceedings IEEE/ACM international conference on Computer-aided design*, pages 158–165, San Jose, California, United States, 1996. IEEE Computer Society. ISBN 0-8186-7597-7. URL http://portal.acm.org/citation.cfm?id=244548&dl=GUIDE&coll=GUIDE&CFID=61193657&CFTOKEN=93601809#.

[11] A. Bogliolo and L. Benini. Robust RTL power macromodels. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4) :578–581, 1998. ISSN 1063-8210.

[12] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. Characterization-free behavioral power modeling. In *Proceedings of the conference on Design, automation and test in Europe*, pages 767–773, 1998.

[13] A. Bogliolo, I. Colonescu, E. Macii, and M. Poncino. An RTL power estimation tool with on-line model building capabilities. *Proc. Int. Wkshp. Power and Timing Modeling, Optimization and Simulation*, pages 391–396, 2001.

[14] Maurizio Bruno, Alberto Macii, and Massimo Poncino. A Statistical Power Model for Non-synthetic RTL Operators. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pages 208–218. Springer, 2003. URL http://www.springerlink.com/content/u1cp3ex26y3kku22.

[15] P. E. Landman and J. M. Rabaey. Architectural power analysis : The dual bit type method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(2) :173–187, 1995. doi : 10.1109/92.386219.

[16] Matthieu Denoual. *Estimation au niveau Architectural de la Consommation des Circuits Dédiés au Traitement Numérique du Signal*. PhD thesis, Université de Rennes 1, 2001.

[17] Subodh Gupta and Farid N. Najm. Power macromodeling for high level power estimation. In *Proceedings of the 34th annual Design Automation Conference*, pages 365–370, Anaheim, California, United States, 1997. ACM. ISBN 0-89791-920-3. doi : 10.1145/266021.266171. URL http://portal.acm.org/citation.cfm?id=266171.

[18] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, and G. De Micheli. Lookup table power macro-models for behavioral library components. In *Proceedings IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 173–181, 1999. doi : 10.1109/LPD.1999.750418.

[19] Roberto Corgnati, Enrico Macii, and Massimo Poncino. Clustered Table-Based Macromodels for RTL Power Estimation. In *Proceedings of the Ninth Great Lakes Symposium on VLSI*, pages 354–357. IEEE Computer Society, 1999. ISBN 0-7695-0104-4. URL http://portal.acm.org/citation.cfm?id=797310.

[20] S. Gupta and F.N. Najm. Analytical model for high level power modeling of combinational and sequential circuits. In *Proceedings IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 164 –172, mar 1999. doi : 10.1109/LPD.1999.750417.

[21] Li Shang and N.K. Jha. High-level power modeling of CPLDs and FPGAs. In *Proceedings International Conference on Computer Design, ICCD*, pages 46–51, 2001.

[22] M. Anton, I. Colonescu, E. Macii, and M. Poncino. Fast characterization of rtl power macromodels. In *Proceedings IEEE the 8th International Conference on Electronics, Circuits and Systems, ICECS*, volume 3, pages 1591–1594, 2001. doi : 10.1109/ICECS.2001.957521.

[23] G. Jochens, L. Kruse, E. Schmidt, and W. Nebel. A new parameterizable power macro-model for datapath components. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, pages 29–36, 1999. doi : 10.1109/DATE.1999.761093.

[24] F. Klein, R. Leao, G. Araujo, L. Santos, and R. Azevedo. A Multi-Model Engine for High-Level Power Estimation Accuracy Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(5) :660–673, 2009. doi : 10.1109/TVLSI.2009.2013627.

[25] P. Surti and L.-F. Chao. Controller power estimation using information from behavioral description. In *Proceedings IEEE International Symposium on Circuits and Systems ISCAS, 'Connecting the World'*, volume 4, pages 679–682, 1996. doi : {10.1109/ISCAS.1996.542115}.

[26] Luca Benini, Alessandro Bogliolo, Enrico Macii, Massimo Poncino, and Mihai Surmei. Regression-based RTL power models for controllers. In *Proceedings of the 10th Great Lakes symposium on VLSI*, pages 147–152, Chicago, Illinois, United States, 2000. ACM. ISBN 1-58113-251-4. doi : 10.1145/330855.331025. URL http://portal.acm.org/citation.cfm?id=331025.

[27] N. Abdelli, A.-M. Fouilliart, N. Mien, and E. Senn. High-Level Power Estimation of FPGA. In *Proceedings IEEE International Symposium on Industrial Electronics, ISIE*, pages 925–930, 2007. doi : 10.1109/ISIE.2007.4374721.

[28] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27 :473 –484, 1992. ISSN 0018-9200.

[29] L. Benini, P. Siegel, and G. De Micheli. Saving power by synthesizing gated clocks for sequential circuits. *IEEE Design Test of Computers*, 11(4) :32 –41, winter 1994. ISSN 0740-7475. doi : 10.1109/54.329451.

[30] Hailin Jiang, M. Marek-Sadowska, and S.R. Nassif. Benefits and costs of power-gating technique. In *Proceedings IEEE International Conference on Computer Design VLSI in Computers and Processors, ICCD*, 2005.

[31] S. Kim, S.V. Kosonocky, and D.R. Knebel. Understanding and minimizing ground bounce during mode transition of power gating structures. In *Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED*, pages 22–25. ACM, 2003.

[32] Wonyoung Kim, M.S. Gupta, Gu-Yeon Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings IEEE 14th International Symposium on High Performance Computer Architecture, HPCA*, pages 123 –134, feb. 2008. doi : 10.1109/HPCA.2008.4658633.

[33] J. Khan, S. Bilavarn, and C. Belleudy. Impact of operating points on DVFS power management. In *Proceedings 7th International Conference on Design Technology of Integrated Systems in Nanoscale Era, DTIS*, pages 1 –6, may 2012. doi : 10.1109/DTIS.2012.6232960.

[34] R. Min, T. Furrer, and A. Chandrakasan. Dynamic voltage scaling techniques for distributed microsensor networks. In *Proceedings IEEE Computer Society Workshop on VLSI*, pages 43–46, 2000.

[35] Altera Corporation. AN531 : Reducing Power with Hardware Accelerators. Technical report, 2008.

[36] Xilinx, Inc. Xilinx XC6200 FPGA Family Data Sheet. Technical report, 1995.

[37] Xilinx. Virtex-II Pro Platform FPGA User Guide. Technical report, 2002.

[38] V. Degalahal and T. Tuan. Methodology for high level estimation of FPGA power consumption. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, volume 1, pages 657–660, 2005. doi : 10.1109/ASPDAC.2005.1466245.

[39] Xilinx Inc. UG440 - Xilinx Power Estimator. Technical report, 2012.

[40] L. Sterpone, L. Carro, D. Matos, S. Wong, and F. Fakhar. A new reconfigurable clock-gating technique for low power SRAM-based FPGAs. In *Proceedings Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1–6. IEEE, 2011.

[41] Qiang Wang, Subodh Gupta, and Jason H. Anderson. Clock power reduction for virtex-5 FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 13–22, 2009. ISBN 978-1-60558-410-2.

[42] Katarina Paulsson, Michael Hubner, and Jurgen Becker. Dynamic power optimization by exploiting self-reconfiguration in Xilinx Spartan 3-based systems. *Microprocessors and Microsystems*, 33 :46 – 52, 2009. ISSN 0141-9331.

[43] Pao-Ann Hsiung and Chih-Wen Liu. Exploiting Hardware and Software Low power Techniques for Energy Efficient Co-Scheduling in Dynamically Reconfigurable Systems. In *Proceedings International Conference on Field Programmable Logic and Applications, FPL*, pages 165–170, 2007.

[44] T.T.-O. Kwok and Yu-Kwong Kwok. Practical design of a computation and energy efficient hardware task scheduler in embedded reconfigurable computing systems. In *Proceedings IEEE 20th International Symposium on Parallel and Distributed Processing, IPDPS*, 2006.

[45] H. Kalte and M. Porrmann. Context saving and restoring for multitasking in reconfigurable systems. In *Proceedings IEEE International Conference on Field Programmable Logic and Applications*, pages 223–228, 2005. doi : 10.1109/FPL.2005.1515726.

[46] Ping-Hung Yuh, Chia-Lin Yang, Chi-Feng Li, and Chung-Hsiang Lin. Leakage-aware task scheduling for partially dynamically reconfigurable FPGAs. *ACM Transactions on Design Automation of Electronic Systems, TODAES*, 14(4) :1–26, 2009. doi : 10.1145/1562514.1562520. URL http://portal.acm.org/citation.cfm?id=1562520&dl=GUIDE&coll=GUIDE&CFID=66627866&CFTOKEN=61412262.

[47] Shaoshan Liu, Richard Neil Pittman, and Alessandro Forin. Energy reduction with run-time partial reconfiguration. In *Proceedings ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 292–292, 2010. ISBN 978-1-60558-911-4.

[48] K.K. Parhi. *VLSI Digital Signal Processing Systems : Design and Implementation*. 1999.

[49] Betul Buyukkurt, Zhi Guo, and Walid Najjar. Impact of Loop Unrolling on Area, Throughput and Clock Frequency in ROCCC : C to VHDL Compiler for FPGAs. In *Reconfigurable Computing : Architectures and Applications*, pages 401–412. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-36708-6.

[50] Scott Thibault and David Pellerin. Optimizing Impulse C Code for Performance. Technical report, Impulse Accelerated Technologies, Inc., 2004. URL http://www.impulseaccelerated.com/AppNotes/.

[51] Catapult C Synthesis, 2013. URL http://calypto.com/en/products/catapult/overview.

[52] Xilinx, Inc. UG902 - Vivado Design Suite User Guide, 2012.

[53] Qiang Liu, T. Todman, and W. Luk. Combining optimizations in automated low power design. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE*, pages 1791–1796, 2010.

[54] Srikanth Kurra, Neeraj Kumar Singh, and Preeti Ranjan Panda. The Impact of Loop Unrolling on Controller Delay in High Level Synthesis. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE*, pages 1–6, 2007.

[55] James G. Eldredge and Brad L. Hutchings. Run-Time Reconfiguration : A Method for Enhancing the Functional Density of SRAM-based FPGAs. *Journal of VLSI signal processing systems for signal, image and video technology*, 12(1) : 67–86, 1996.

[56] Kizheppatt Vipin and SuhaibA. Fahmy. Architecture-Aware Reconfiguration-Centric Floorplanning for Partial Reconfiguration. In OliverC.S. Choy, RayC.C. Cheung, Peter Athanas, and Kentaro Sano, editors, *Reconfigurable Computing : Architectures, Tools and Applications*, volume 7199 of *Lecture Notes in Computer Science*, pages 13–25. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28364-2. doi : 10.1007/978-3-642-28365-9_2. URL http://dx.doi.org/10.1007/978-3-642-28365-9_2.

[57] François Duhem, Fabrice Muller, and Philippe Lorenzini. Methodology for Designing Partially Reconfigurable Systems Using Transaction-Level Modeling. In *Proceedings of the International Conference on Design and Architectures for Signal and Image Processing, DASIP*, pages 1–7, 2011.

[58] Kyprianos Papadimitriou, Apostolos Dollas, and Scott Hauck. Performance of partial reconfiguration in FPGA systems : A survey and a cost model. *ACM*

Transactions on Reconfigurable Technology and Systems, TRETS, 4(4) :36 :1–36 :24, 2011. ISSN 1936-7406.

[59] Xilinx, Inc. DS586 LogiCORE IP XPS HWICAP. Technical report, 2010.

[60] Xilinx, Inc. UG081 MicroBlaze Processor Reference Guide. Technical report, 2011.

[61] Xilinx, Inc. PowerPC 405 Processor Block Reference Guide. Technical report, 2004.

[62] Ming Liu, W. Kuehn, Zhonghai Lu, and A. Jantsch. Run-time Partial Reconfiguration speed investigation and architectural design space exploration. In *Proceedings of the International Conference on Field Programmable Logic and Applications, FPL*, pages 498 –502, 31 2009-sept. 2 2009. doi : 10.1109/FPL.2009.5272463.

[63] Atukem Nabina and Jose L. Nunez-Yanez. Dynamic Reconfiguration Optimisation with Streaming Data Decompression. In *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications, FPL*, pages 602–607, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4179-2. doi : http://dx.doi.org/10.1109/FPL.2010.118. URL http://dx.doi.org/10.1109/FPL.2010.118.

[64] José Luis Núñez and Simon Jones. Gbits/s Lossless Data Compression Hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11 : 499–510, 2003.

[65] Duhem, F. and Muller, F. and Lorenzini, P. FaRM : Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA. In *Reconfigurable Computing : Architectures, Tools and Applications*, pages 253–260. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-19474-0.

[66] F. Duhem, F. Muller, and P. Lorenzini. Reconfiguration time overhead on field programmable gate arrays : reduction and cost model. *Computers & Digital Techniques, IET*, 6 :105–113, 2012.

[67] T. Becker, W. Luk, and P. Y. K. Cheung. Energy-Aware Optimisation for Run-Time Reconfiguration. In *Proceedings of the IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM*, pages 55–62, 2010.

[68] J. Becker, M. Huebner, and M. Ullmann. Power estimation and power measurement of Xilinx Virtex FPGAs : trade-offs and limitations. In *Proceedings of the 16th IEEE Symposium on Integrated Circuits and Systems Design, SBCCI*, pages 283–288, 2003.

[69] Savary Yannig. *Etude du potentiel des architectures reconfigurables pour maîtriser la consommation dans les applications embarquées.* PhD thesis, Université de bretagne sud - Ecole doctorale Pluridisciplinaire, 2007.

[70] Michael Lorenz, Luis Mengibar, Mario Valderas, and Luis Entrena. Power Consumption Reduction Through Dynamic Reconfiguration. In *Field Programmable Logic and Application*, pages 751–760. Springer Berlin / Heidelberg, 2004.

[71] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka. Dynamic and Partial FPGA Exploitation. *Proceedings of the IEEE*, 95(2) : 438–452, 2007.

[72] UML - Unified Modeling Language, 2013. URL http://www.uml.org/.

[73] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A SoC design methodology involving a UML 2.0 profile for SystemC. In *Proceedings of the International Conference on Design, Automation and Test in Europe, DATE*, pages 704 – 709, march 2005. doi : 10.1109/DATE.2005.37.

[74] OMG Systems Modeling Language, 2013. URL http://www.omgsysml.org/.

[75] The UML Profile for MARTE : Modeling and Analysis of Real-Time and Embedded Systems, 2013. URL http://www.omgmarte.org/.

[76] I.R. Quadri, S. Meftali, and J.-L. Dekeyser. Designing dynamically reconfigurable SoCs : From UML MARTE models to automatic code generation. In *Proceedings of the International Conference on Design and Architectures for Signal and Image Processing, DASIP*, pages 68 –75, oct. 2010. doi : 10.1109/DASIP.2010.5706248.

[77] M.A. Peraldi-Frati and Y. Sorel. From high-level modelling of time in MARTE to real-time scheduling analysis. *International Workshop on Model Based Architecting and Construction of Embedded Systems, ACESMB*, page 129, 2008.

[78] Detailed presentation of AADL, 2013. URL http://www.axlog.fr/aadl/presentation_en.html.

[79] AADL, 2013. URL http://www.aadl.info/aadl/currentsite/.

[80] Open-PEOPLE - Open-Power and Energy Optimization PLatform and Estimator, 2013. URL http://www.open-people.fr/.

[81] Xiaoheng Chen and Venkatesh Akella. Exploiting data-level parallelism for energy-efficient implementation of LDPC decoders and DCT on an FPGA. *ACM Transactions on Reconfigurable Technology and Systems, TRETS*, 4(4) : 37 :1–37 :17, December 2011. ISSN 1936-7406. doi : 10.1145/2068716.2068723. URL http://doi.acm.org/10.1145/2068716.2068723.

[82] T. Damak, I. Werda, N. Masmoudi, and S. Bilavarn. Fast prototyping H.264 Deblocking filter using ESL tools. In *Proceedings of the IEEE 8th International Multi-Conference on Systems, Signals and Devices, SSD*, pages 1–4, 2011.

[83] Xilinx, Inc. UG744 - PlanAhead Software Tutorial : Partial Reconfiguration of a Processor Peripheral. Technical report, 2011.

[84] Xilinx, Inc. UG191 - Virtex-5 FPGA Configuration User Guide. Technical report, 2010.

[85] Hung-Manh Pham, Van-Cuong Nguyen, and Trong-Tuan Nguyen. DDR2/DDR3-based ultra-rapid reconfiguration controller. In *Proceedings of the IEEE Fourth International Conference on Communications and Electronics, ICCE*, pages 453–458, 2012. doi : 10.1109/CCE.2012.6315949.

[86] Xilinx, Inc. LogiCORE IP Block Memory Generator v4.3. Technical report, 2008.

[87] Xilinx, Inc. UG347 – ML505/ML506/ML507 Evaluation Platform. Technical report, 2008.

[88] Xilinx, Inc. Virtex-5 FPGA User-Guide. Technical report, January 2009.

[89] Xilinx, Inc. UG534 – ML605 Hardware User Guide. Technical report, 2011.

[90] Xilinx, Inc. UG360 – Virtex-6 FPGA Configuration User Guide (v3.1). Technical report, July 2010.

[91] François Duhem, Fabrice Muller, Willy Aubry, Bertrand Le Gal, Daniel Négru, and Philippe Lorenzini. Design Space Exploration for Partially Reconfigurable Architectures in Real-Time Systems. *Journal of Systems Architecture (JSA)*, Under review (2013).

[92] ISO/IEC 14496-10, Advanced Video Coding for Generic Audiovisual Services, ITU-T Recommendation H.264, Version 4, 2005.

[93] R. Urunuela, A. Deplanche, and Y. Trinquet. STORM a simulation tool for real-time multiprocessor scheduling evaluation. In *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2010.

[94] STORM : Simulation TOol for Real time Multiprocessor scheduling, 2013. URL http://storm.rts-software.org/doku.php.